RECORD AND BRIEFS

Supreme Court, U.S.
FILED
JUL 20 2017
OFFICE OF THE CLERK

No. 16-969

IN THE
Supreme Court of the United States

SAS INSTITUTE INC.,
Petitioner,

v.

JOSEPH MATAL, Interim Director, U.S. Patent and Trademark Office, and COMPLEMENTSOFT, LLC,
Respondents.

On Writ of Certiorari to the United States Court of Appeals for the Federal Circuit

JOINT APPENDIX VOLUME I OF II

GREGORY A. CASTANIAS
Counsel of Record
JONES DAY
51 Louisiana Ave., NW
Washington, DC 20001
(202) 879-3939
gcastanias@jonesday.com

Counsel for Petitioner

JEFFREY B. WALL
Counsel of Record
DEPARTMENT OF JUSTICE
950 Pennsylvania Ave., NW
Washington, DC 20530
(202) 514-2217
supremectbriefs@usdoj.gov

Counsel for Respondents

PETITION FOR CERTIORARI FILED JANUARY 31, 2017
CERTIORARI GRANTED MAY 22, 2017

TABLE OF CONTENTS

Page

Docket Entries, *SAS Institute v. ComplementSoft*, No. IPR2013-00226 (USPTO) 1

Petition for *Inter Partes Review* 4

Order Granting Certiorari 79

U.S. DEPARTMENT OF COMMERCE
United States Patent and Trademark Office

February 18, 2015
(Date)

THIS IS TO CERTIFY that the attached document is a list of the papers that comprise the record before the Patent Trial and Appeal Board (PTAB) for the *Inter Partes Review* proceeding identified below.

SAS INSTITUTE, INC.
Petitioner,
v.
COMPLEMENTSOFT, LLC,
Patent Owner
Case: IPR2013-00226
Patent No. 7,110,936 B2

By authority of the
DIRECTOR OF THE UNITED STATES
PATENT AND TRADEMARK OFFICE

/s/Kyra Abraham
Certifying Officer

Prosecution History IPR2013-00226

Date	Document
3/29/2013	**Petition for Inter Partes Review**
3/29/2013	Power of Attorney
4/5/2013	Notice of Filing Date Accorded to Petition
4/24/2013	Patent Owner's Power of Attorney
4/24/2013	Patent Owner's Power of Attorney
4/25/2013	Patent Owner's Mandatory Disclosures
5/31/2013	Petitioner's Revised Mandatory Notices
6/28/2013	Patent Owner's Preliminary Response
8/12/2013	Decision - Institution of Inter Partes Review
8/12/2013	Scheduling Order
9/9/2013	Patent Owner's Proposed Motions List
9/13/2013	Order - Conduct of the Proceeding
9/20/2013	Order - Conduct of the Proceeding
10/18/2013	Notice of Deposition - Roussopoulos
11/7/2013	Order - Conduct of the Proceeding
11/12/2013	Patent Owner's Response to Petition
12/17/2013	Order - Conduct of the Proceeding
12/20/2013	Substitute Contingent Motion to Amend
12/23/2013	Notice of Deposition - Hiew
12/23/2013	Notice of Deposition -Zatkovich
1/14/2014	Petitioner's Second Deposition Notice - Hiew
2/12/2014	Reply to Patent Owner's Response

Date	Document
2/12/2014	Opposition to Patent Owner's Motion to Amend
3/12/2014	Reply for Contingent Motion to Amend
3/20/2014	Order - Conduct of the Proceeding
4/2/2014	Petitioner's Request for Oral Hearing
4/4/2014	Order - Trial Hearing
5/5/2014	Order - Conduct of the Proceeding
5/5/2014	Petitioner's Submission of Demonstrative Exhibits
5/5/2014	Petitioner's Updated Exhibit List
5/5/2014	Patent Owner's Submission of Demonstrative Exhiibits
5/5/2014	Patent Owner's Updated Exhibit List
6/4/2014	Record of Oral Hearing
6/13/2014	Power of Attorney
8/6/2014	Final Written Decision
9/5/2014	Petitioner's Request for Rehearing
11/10/2014	Decision - Request for Rehearing

IN THE UNITED STATES PATENT TRIAL AND APPEAL BOARD

In the *Inter Partes* Review of U.S. Patent No. 7,110,936

Trial No.: Not Yet Assigned

Issued: September 19, 2006

Filed: November 19, 2001

Inventors: Fen Hiew, *et al.*

Assignee: Complementsoft LLC

Title: SYSTEM AND METHOD FOR GENERATING AND MAINTAINING SOFTWARE CODE

MAIL STOP *PATENT BOARD*
Patent Trial and Appeal Board
United States Patent & Trademark Office
P.O. Box 1450
Alexandria, Virginia 22313-1450

PETITION FOR *INTER PARTES* REVIEW UNDER 37 C.F.R. § 42.100

On behalf of SAS Institute, Inc. ("SAS" or "Petitioner") and in accordance with 35 U.S.C. § 311 and 37 C.F.R. § 42.100, *inter partes* review is respectfully requested for claims 1-16 of U.S. Patent No. 7,110,936 ("the '936 patent"), attached hereto as Exhibit 1001.

The undersigned representative of Petitioner authorizes the Patent Office to charge the $27,200 Petition Fee, along with any additional fees, to Deposit Account 501432, ref: 343355-610009. Sixteen claims are being reviewed; accordingly no excess claim fees are required.

Table of Contents

I. Introduction 1

II. Grounds for Standing Pursuant to 37 C.F.R. § 42.104(a) 5

III. The '936 Patent 5

A. Summary of the '936 Patent 5

B. The '936 Patent Prosecution History .. 7

IV. Identification of Challenge Pursuant to 37 C.F.R. § 42.104(b) 10

A. 37 C.F.R. § 42.104(b)(1): Claims For Which *Inter Partes* Review Is Requested 10

B. 37 C.F.R. § 42.104(b)(2): The Prior Art and Specific Grounds On Which The Challenge to the Claims Is Based 10

C. 37 C.F.R. § 42.104(b)(3): Claim Construction 12

D. 37 C.F.R. § 42.104(b)(4): How the Construed Claims are Unpatentable 14

E. 37 C.F.R. § 42.104(b)(5): Supporting Evidence 14

V. There Is a Reasonable Likelihood That at Least One Claim of the '936 Patent Is Unpatentable 15

A. Overview of the Prior Art 15

B. Claim 1 Is Anticipated by Coad (U.S. Patent No. 6,851,107)..... 25

C. Claim 1 Is Obvious Over Coad in View of Oracle Primer and Oracle8 Primer ... 31

D. Claims 1-3 and 5 Are Anticipated by Antis (U.S. Patent No. 5,572,650) 32

E. Claims 1-3, 5, 6, 8, 10-12, 15, and 16 Are Obvious Over Antis in View of Coad. 41

F. Claim 4 Is Obvious Over Antis in View of Coad and Burkwald (U.S. Patent No. 6,356,285) 52

G. Claim 7 Is Obvious Over Antis in view of Coad and Eick (U.S. Patent No. 5,937,064) 53

H. Claim 9 Is Obvious Over Antis in view of Coad and "Building Applications" 54

I. Claim 13 Is Obvious Over Antis in view of Coad and Corda (U.S.Patent No. 5,758,122) 56

J. Claim 14 Is Obvious Over Antis in view of Coad and Access 97 Visual Basic .. 57

K. Dependent Claims 2-16 Are Unpatentable Over Each Reference and Combination of References Asserted for Claim 1 in View of the Prior Art..... 57

VI. Mandatory Notices Pursuant to 37 C.F.R. § 42.8(a)(1)..... 58

A. C.F.R. § 42.8(b)(a): Real Party-In-Interest 58

B. C.F.R. § 42.8(b)(2): Related Matters.. 58

C. C.F.R. § 42.8(b)(3) and (4): Lead and Back-up Counsel and Service Information..... 58

VII. Conclusion 59

I. Introduction

The '936 patent is currently being wielded by the patent owner, Complementsoft LLC ("Complementsoft"), in an attempt to cover long-known systems and methods for developing software. (*ComplementSoft, LLC v. SAS Institute Inc.*, Docket No. 1:12-cv-07372 (N.D. Ill. Sept 14, 2012).) The '936 patent is directed to an integrated development environment for generating and maintaining source code and discloses four elements that work in concert – a document manager, an editor, a parser layer, and a visualizer. (*See, e.g.*, '936 patent, claim 1.) The subject matter claimed in the '936 patent includes standard elements that were well known in the prior art before the filing date of the '936 patent, and this was acknowledged by the Office.[1] Complementsoft itself has conceded that numerous claim elements of the '936 patent were disclosed in the prior art, as evidenced in documents filed in the related litigation. (*See* Ex. 1016, Complementsoft's LPR 2.5 Initial Response to SAS Institute's Invalidity Contentions.)

---

[1] In a Supplemental Notice of Allowability for the '936 patent, the Office stated:

> "Applicants are disclosing a software development environment where a user can simultaneously view a graphical representation and a text representation of source code including views that are synchronized such that modifications to one view are automatically reflected in the other view, including the ability to detect the particular language of the source code by applying rules. This has been disclosed in the prior art of record." (Ex. 1004, Supplemental Notice of Allowability dated August 1, 2006, at 3-4.)

In fact, during prosecution of the '936 patent, the Office found that *all* four elements of claim 1 – the document manager, editor, parser layer, and visualizer – were disclosed in a single reference and in the same arrangement as claimed in the '936 patent. (Ex. 1002, Office Action dated May 31, 2005, at 3, noting "[The Coad reference] clearly anticipates the claimed limitations of independent claims 1 and 18.")

Despite this finding, however, Complementsoft was able to gain allowance based on a single aspect that allegedly distinguished the prior art from the pending claims. During prosecution, Complementsoft attempted to distinguish the prior art, which was applicable to object-oriented languages, from the amended claims, that were modified to recite data manipulation languages.[2] Complementsoft argued that the object-oriented languages used in the prior art were "structured" programming languages, while the data manipulation languages of the amended claims were "unstructured" programming languages, and that the claims were thus distinguishable from the prior art on this basis.

Complementsoft's argument, however, is technically misleading because a person of ordinary skill in the art would understand that most data manipulation languages are in fact structured programming languages, and the claims should not have been allowed on this basis. (*See* Ex. 1015, ¶¶ 24-60.) Further, the scope of claim 1, covering both program flows *and* data flows, is inconsistent with

---

[2] As filed, the claims were directed to any form of "source code."

the argument set forth by Complementsoft. This is because the distinction argued by Complementsoft (*i.e.*, that displaying flow in unstructured programming languages is distinguishable from displaying flow in structured programming languages) applies, if at all, to displaying *program flows* only. (*Id. at* ¶¶ 61-68.)

Moreover, as described below, the claimed software development environment was not new when the '936 patent was filed. As noted in the background of the '936 patent, GUI-based software editor packages are very old, and the problems inherent with such software were common knowledge in the field: "There are also a few GUI based editor software packages to make editing code easier.... What is lacking, however, is [an] . . . editor integrated with [a] diagramming package so the user does not have to manually generate and update program flow or data flow diagrams." ('936 patent at col. 1, lines 52-63.) The '936 patent purports to address these problems by dynamically linking an editor and a visualizer, such that edits made to source code using the editor are automatically reflected in graphical representations of flows displayed by the visualizer, and vice versa. (*See, e.g.*, *id. at* claim 1.) But the linking of an editor and a visualizer was a common component of software development environments at the time that the '936 patent was filed.

Accordingly, the lack of inventiveness is evidenced herein by the refutation of the argument presented during prosecution with respect to Coad. The lack of inventiveness is further shown, *inter alia*, by the submission of U.S. Patent No. 5,572,650 (Ex. 1005, "Antis"), which issued more than five years prior to

the filing of the '936 patent in November 2001. Antis discloses all of the limitations of the independent claim of the '936 patent, including the visualization of flows within source code programmed using a data manipulation language. Specifically, Antis discloses a "method and apparatus for visually displaying structural characteristics of a large database." (Antis at Abstract.) Antis discloses the display of graphical representations of flows in source code, and, in particular, flows in *data manipulation language* source code. In fact, the types of flows disclosed in the Antis patent are *nearly identical* to those described and claimed in the '936 patent. (*See, e.g.*, Fig. 17 of the '936 patent, disclosing flow by using arrows to display the data that is accessed by retrieved pieces of source code, and Antis at col. 7, line 31 to col. 8, line 4, disclosing a "code view" that provides a visual representation of the database relations that are accessed by a selected piece of source code.)

The dependent claims of the '936 patent add nothing more than well known software editing concepts that are explicitly disclosed in the Coad reference applied during prosecution or in one or more additional prior art references that are analogous art to Antis and that one of ordinary skill in the art would be motivated to combine with Antis in generating an integrated software development environment.

Petitioner submits that had the references presented herein been properly considered by the Patent Office during prosecution, claims 1-16 of the '936 patent would not have issued, and therefore this petition for *inter partes* review should be granted.

II. Grounds for Standing Pursuant to 37 C.F.R. § 42.104(a)

SAS certifies that the '936 patent is available for *inter partes* review and that SAS is not barred or estopped from requesting *inter partes* review challenging the patent claims on the grounds identified herein.

III. The '936 Patent

A. Summary of the '936 Patent

The '936 patent was filed on November 19, 2001, and issued on September 19, 2006. The '936 patent claims priority to U.S. Provisional Application Nos. 60/270,950 and 60/293,854, filed on February 23, 2001, and May 25, 2001, respectively.

The '936 patent is directed to an integrated software development environment for generating and editing computer code. ('936 patent at Abstract.) The integrated software development environment includes four modules: (1) a document manager, (2) an editor, (3) a parser layer, and (4) a visualizer. (*See id.* at col. 5, lines 4-25.) The document manager is a file management program used to retrieve source code for editing. (*Id.* at col. 6, lines 22-42.) The retrieved source code is edited using the editor module, which "allows the user to perform standard text editing functions, including, mouse placement of the cursor, click-and-drag text selection and standard Windows key combinations for cutting, copying and pasting data." (*Id.* at col. 7, lines 4-7.)

The parser layer and the visualizer are used to display graphical representations of flows within the retrieved source code. The parser layer is "language aware," such that the integrated software

development environment of the '936 patent is able to develop and edit code of multiple different programming languages. (*Id.* at col. 17, lines 17-32; *see also id.* at col. 9, lines 49-53, disclosing use with SAS, SQL, SPSS, DB2 UDB, Oracle, and PL/SQL languages.) Using the rules and logic that correspond to the detected programming language, the file parser breaks the retrieved source code into individual words or tokens. (*Id.* at col. 17, lines 46-49.) The file parser also tags each token to associate it with a particular class. (*Id.* at col. 17, lines 48-54.)

Tagging of the tokens enables the visualizer module to recognize and display the flow of the retrieved source code. (*Id.* at col. 17, lines 54-56.) Fig. 17, for example, is an exemplary screenshot from the visualizer module depicting a flow of retrieved source code. In Fig. 17, the visualizer module displays at 120 icons 126 with arrows that illustrate data flow in the retrieved source code. The flow thus provides an indication of what data (*e.g.*, relations smpl.smpl_1, smpl.smpl_2) are accessed by which modules of source code (*e.g.*, source code used to perform a sort operation).

The visualizer module is dynamically linked with the editor module, such that edits made to the source code using the editor are automatically reflected in the graphical representations of flows displayed by the visualizer, and edits made to the graphical representations of flows in the visualizer are automatically reflected in the source code displayed by the editor. (*See, e.g.*, *id.* at claim 1 and col. 16, lines 31-46.)

B. The '936 Patent Prosecution History

The '936 patent was allowed after responses to two non-final Office Actions were filed. The first non-final Office Action rejected the original claims based upon the Coad reference (Ex. 1006, U.S. Patent No. 6,851,107). The Office found that all four elements of independent claim 1 were disclosed in the Coad reference and in the same arrangement as claimed in the '936 patent. (Ex. 1002, Office Action dated May 31, 2005, at 2-3, noting that "Coad clearly anticipates the claimed limitations of independent claims 1 and 18.") In an attempt to distinguish the pending claims from Coad, Complementsoft amended independent claim 1 as follows:

> 1. (Currently Amended) An integrated development environment, comprising:
>
> a document manager for retrieving source code programmed using one of a plurality of types of data manipulation languages;
>
> an editor for displaying the retrieved source code and providing a means for a user to edit the retrieved source code;
>
> a parser layer which detects the one of the plurality of types of data manipulation languages in which the ~~software language of the~~ retrieved source code is programmed and which activates rules and logic applicable to the detected one of the plurality of types of data manipulation languages ~~software language~~; and
>
> a visualizer dynamically linked to the editor for displaying graphical representations of flows within the retrieved source code using the rules and logic applicable to the detected one of the

> plurality of types of data manipulation languages and activated by the parser, wherein the editor, parser layer and visualizer cooperate such that edits made to the source code using the editor are automatically reflected in the graphical representations of flows displayed by the visualizer and edits made to the graphical representations of flows in the visualizer are automatically reflected in the source code displayed by the editor.

In its remarks, Complementsoft did not dispute that all four elements of the independent claim (*i.e.*, the document manager, editor, parser, and visualizer) were disclosed in Coad. Instead, Complementsoft argued:

> Considering now Coad, Coad discloses a software development tool for use with object-oriented programming languages, such as Java or C++. More particularly, Coad discloses a system in which the object-oriented language of the source code is identified by the extension of a retrieved file to thereby identify a template which is to be used to convert the source code into a transient meta model (TMM), *i.e.*, a language-neutral representation of the source code. The system then uses the TMM to display a graphical representation of a project.
>
> From the foregoing it will be appreciated that Coad fails to expressly or inherently disclose the claimed system and method. More particularly, rather than disclose the claimed system and method for examining software written using one of the plural **data manipulation**

> **languages, *i.e.*, a language which suffers the problem of being unstructured** (*see* Background section of subject application), the system and method described in Coad is limited to software written using an **object oriented language, *i.e.*, a structured programming language** such as Java or C++. . . . [T]he system and method of Coad leverages the **structured** nature of the programming language to merely "convert" the source code into a language-neutral representation and then use the language-neutral representation to display a graphical representation of the project. (Ex. 1003, Response to Office Action dated August 4, 2005, at 9-10, emphasis added.)

Complementsoft thus argued that the object-oriented languages of Coad were "structured" programming languages, while the data manipulation languages of the claims were "unstructured" programming languages, and that the claims were distinguishable from Coad on this basis.

Claim 1 ultimately issued in the form shown above. A supplemental notice of allowability provided the following with respect to claim 1 overcoming Coad:

> Coad further discloses the ability to detect the particular language of the source code and applying rules. However, Coad does not explicitly disclose the specific arrangement of elements including a document manager, editor, parser layer, or visulizer [sic] as noted above, or that the detected language is a <u>data manipulation language</u>. (Ex. 1004,

Supplemental Notice of Allowability at 5, emphasis in original.)

IV. Identification of Challenge Pursuant to 37 C.F.R. § 42.104(b)

A. 37 C.F.R. § 42.104(b)(1): Claims For Which *Inter Partes* Review Is Requested

Inter partes review is requested for claims 1-16 of the '936 patent.

B. 37 C.F.R. § 42.104(b)(2): The Prior Art and Specific Grounds On Which The Challenge to the Claims Is Based

Inter partes review is requested in view of the following prior art references:

- U.S. Patent No. 5,572,650 to Antis *et al.* ("Antis") (Exhibit 1005). Antis was filed on June 30, 1994, and issued on November 5, 1996. Antis is therefore prior art to the '936 patent under 35 U.S.C. § 102(b).
- U.S. Patent No. 6,851,107 to Coad *et al.* ("Coad") (Exhibit 1006). Coad was filed on October 4, 2000, and issued on February 1, 2005. Coad is therefore prior art to the '936 patent under 35 U.S.C. § 102(e).
- U.S. Patent No. 6,356,285 to Burkwald *et al.* ("Burkwald") (Exhibit 1007). Burkwald was filed on December 17, 1997, and issued on March 12, 2002. Burkwald is therefore prior art to the '936 patent under 35 U.S.C. § 102(e).
- U.S. Patent No. 5,937,064 to Eick *et al.* ("Eick") (Exhibit 1008). Eick was filed on March 3, 1997, and issued on August 10, 1999. Eick is therefore

prior art to the '936 patent under 35 U.S.C. § 102(b).

- "Microsoft Access 97 Visual Basic Step by Step" (hereinafter "Access 97 Visual Basic") (Exhibit 1009). Access 97 Visual Basic was published in 1997. Access 97 Visual Basic is therefore prior art to the '936 patent under 35 U.S.C. § 102(b).
- U.S. Patent No. 5,758,122 to Corda *et al.* ("Corda") (Exhibit 1010). Corda was filed on March 15, 1995, and issued on May 26, 1998. Corda is therefore prior art to the '936 patent under 35 U.S.C. § 102(b).
- "Building Applications with Microsoft Access 97" (hereinafter "Building Applications") (Exhibit 1011). "Building Applications" was published in 1996. "Building Applications" is therefore prior art to the '936 patent under 35 U.S.C. § 102(b).
- "Oracle Programming – A Primer" (hereinafter "Oracle Primer") (Exhibit 1012). Oracle Primer was published in 1999. Oracle Primer is therefore prior art to the '936 patent under 35 U.S.C. § 102(b).
- "Oracle8 Programming: A Primer" (hereinafter "Oracle8 Primer") (Exhibit 1013). Oracle8 Primer was published in 1999. (*See* Ex. 1017.) Oracle8 Primer is therefore prior art to the '936 patent under 35 U.S.C. § 102(b).

The specific statutory grounds on which the challenge to the claims is based and the patents relied upon for each ground are as follows:

a) Claim 1 is anticipated by Coad under 35 U.S.C. § 102(e).

b) Claim 1 is unpatentable under 35 U.S.C. § 103(a) over Coad in view of Oracle Primer and Oracle8 Primer.

c) Claims 1-3 and 5 are anticipated by Antis under 35 U.S.C. § 102(b).

d) Claims 1-3, 5, 6, 8, 10-12, 15, and 16 are unpatentable under 35 U.S.C. § 103(a) over Antis in view of Coad.

e) Claim 4 is unpatentable under 35 U.S.C. § 103(a) over Antis in view of Coad and Burkwald.

f) Claim 7 is unpatentable under 35 U.S.C. § 103(a) over Antis in view of Coad and Eick.

g) Claim 9 is unpatentable under 35 U.S.C. § 103(a) over Antis in view of Coad and "Building Applications."

h) Claim 13 is unpatentable under 35 U.S.C. § 103(a) over Antis in view of Coad and Corda.

i) Claim 14 is unpatentable under 35 U.S.C. § 103(a) over Antis in view of Coad and Access 97 Visual Basic.

j) To the extent not explicitly enumerated above, claims 2-16 are unpatentable over each reference and combination of references asserted for claim 1 in view of the prior art.

C. 37 C.F.R. § 42.104(b)(3): Claim Construction

Pursuant to 37 C.F.R. § 42.100(b), and solely for the purposes of this review, Petitioner construes the claim language such that the claims are given their broadest reasonable interpretation in light of the specification of the '936 patent. For terms not specifically listed and construed below, Petitioner

interprets them for purposes of this review in accordance with their plain and ordinary meaning under the required broadest reasonable interpretation.

"***automatically***" — Claim 1. For purposes of this review, Petitioner adopts the construction "*without user intervention*" for the term "automatically." This is consistent with the '936 patent's specification as well as applicable case law. In the specification, for example, the '936 patent states: "Preferably, the visualizer and editor are integrated so that changes made to the code in the editor are *immediately* reflected in the visualizer and vice-versa." ('936 patent at 2:42-45; emphasis added). The term "automatic" has been construed previously in other litigations involving software technology. *See*, for example: *Collegenet, Inc. v. Applyyourself, Inc.*, 418 F.3d 1225 (Fed. Cir. 2005) (affirming the District Court's interpretation that the claim term "automatically" means "once initiated, the function is performed by a machine, without the need for manually performing the function."); *Eolas Technologies, Inc. v. Adobe Systems, Inc.*, 810 F. Supp.2d 795 (E.D. Texas 2011) (finding that the claim term "automatically invoking the executable applications" meant that the executable application is launched without user activation.); and *Rockwell Electronic Commerce Corporation v. Apropos Technology, Inc.*, 2002 U.S. Dist. LEXIS 272 (N.D. Illinois 2002) (finding that the claim term "automatically" in several phrases means "initiated and performed without human intervention.")

"***data manipulation language***" — Claim 1. For purposes of this review, Petitioner adopts the

construction "*a programming language used to access data in a database, such as to retrieve, insert, delete, or modify data in the database*" for the term "data manipulation language." (*See* Ex. 1015, ¶¶ 48-54.) This is consistent with the '936 patent's specification. In the specification, for example, the '936 patent cites SQL and Oracle as data manipulation languages for accessing data in a database. (*See, e.g.*, '936 patent at col. 1, lines 20-25.) Based on such disclosures, the adopted claim construction for "data manipulation language" is consistent with the disclosure in the '936 patent with respect to data manipulation languages.

Because the standard for claim construction at the Patent Office is different than that used during a U.S. District Court litigation, *see In re Am. Acad. Of Sci. Tech Ctr.*, 367 F.3d 1359, 1364, 1369 (Fed. Cir. 2004); MPEP § 2111, Petitioner expressly reserves the right to argue a different claim construction in litigation for any term of the '936 patent as appropriate in that proceeding.

D. 37 C.F.R. § 42.104(b)(4): How the Construed Claims are Unpatentable

A detailed explanation of how claims 1-16 are unpatentable is set forth below at Section V.

E. 37 C.F.R. § 42.104(b)(5): Supporting Evidence

An Appendix of Exhibits supporting this Petition is attached. Included at Exhibit 1015 is a Declaration of Dr. Nick Roussopoulos under 37 C.F.R. § 1.68. References discussed in the Declaration of Dr. Nick Roussopoulos are attached as Exhibits 1021 – 1040.

V. There Is a Reasonable Likelihood That at Least One Claim of the '936 Patent Is Unpatentable

Section V.A contains a description of each piece of relevant prior art and an explanation of the motivation to combine the relevant prior art (where applicable), and Section V.B contains claim charts identifying where each element of the challenged claims are found in the prior art.

A. Overview of the Prior Art

1. Coad

"Coad" (Ex. 1006), U.S. Patent No. 6,851,107, was filed on October 4, 2000, and issued on February 1, 2005. Coad is therefore prior art to the '936 patent under 35 U.S.C. § 102(e). Coad was applied against the claims of the '936 patent during prosecution. (*See* Ex. 1002, Office Action dated May 31, 2005.) During prosecution, the Office found that Coad disclosed numerous limitations of the claims, including "a visualizer dynamically linked to the editor for displaying graphical representations of flows within the retrieved source code." (*See id. at* 3.)

Coad is directed to a "software development tool which allows a developer to simultaneously view a graphical and a textual display of source code." (Coad at Abstract.) The graphical and textual views are synchronized, such that modifications made in one view are automatically reflected in the other view. (*Id.* at col. 4, line 64 to col. 5, line 2.) To display the graphical view of the source code, the software development tool of Coad "determines the language of the source code of the chosen file, converts the source code from the language into a

language-neutral representation, uses the language-neutral representation to textually display the source code of the chosen file in the language, and uses the language-neutral representation to display a graphical representation of at least a portion of the project." (*Id.* at col. 15, lines 21-28.) The graphical view of the source code includes graphical representations of flows within the source code. (*Id.* at col. 16, line 57 to col. 17, line 47.)

The software development tool of Coad further discloses a Quality Assurance (QA) module used to check source code for various types of errors (*i.e.*, coding errors, critical errors, and declaration style errors, among others). (*Id.* at col. 9, line 66 to col. 10, line 5.) If errors are detected in the source code, an error message and debugging tip for correcting the code are provided to the developer. (*Id.* at col. 14, line 64 to col. 15, line 15.)

2. Antis

"Antis" (Ex. 1005), U.S. Patent No. 5,572,650, was filed on June 30, 1994, and issued on November 5, 1996. Antis is therefore prior art to the '936 patent under 35 U.S.C. § 102(b).

Antis is directed to a relational database analysis system that provides graphical displays for visualizing structures and relationships within relational databases. (Antis at col. 1, lines 15-18.) To provide the visualizations, the relational database analysis system includes a number of linked "views." For example, a highest level view of a relational database is presented in an "over view," as illustrated in Fig. 2 of Antis. The over view displays high-level statistics and characteristics of the relational

database including a listing of all relations of the database. (*Id.* at col. 4, lines 1-19.)

Further information on the relational database is provided in additional views that are interactively linked to the over view and to each other. (*Id.* at col. 2, lines 36-39; *see also id.* at Figs. 2-12, depicting linked views including specification view, associations view, code view, and domain view, among others.) For example, in Fig. 8, Antis discloses a "code view" including a plurality of code boxes, where each code box represents a unit of source code of a relational database management system (RDBMS). The code view is linked to the over view, such that associations between relations of the database and the source code that accesses the relations can be visually represented. For example, if a relation is selected in the over view, the application source code that "use[s] the currently selected relation [is] displayed in the code view." (*Id.* at col. 7, lines 36-38.) Similarly, "[c]licking the mouse button on a code box [in the code view] highlights in the over view all relations that use the corresponding unit of code." (*Id.* at col. 7, lines 52-54.) The associations between the relations of the database and the source code that accesses the relations are determined via a parser layer that is based on the YACC computer program. (*Id.* at col. 9, lines 29-41.) The parser layer detects one of a plurality of different types of programming languages used in source code and applies appropriate rules and logic based on the detected language. (*Id.*)

The linking of the various views "makes them interactive with each other, such that changes to one view causes corresponding changes in the other

actively linked views." (*Id.* at col. 9, lines 23-25.) For example, Antis discloses that a piece of source code can be retrieved via Cscope, "a well known code browser." (*Id.* at col. 9, line 46.) Any edits to the retrieved source code "will have coordinated interactive results in the other open views." (*See id.* at col. 8, line 63 to col. 9, line 1, and Fig. 12.)

3. Motivation to Combine Antis with Coad

A person of ordinary skill in the art (POSITA) would have been motivated to combine Antis's teachings with Coad because both Antis and Coad are directed to software development tools that provide visual representations of source code. The similar purposes and overlapping teachings that would have motivated the POSITA to combine the teachings of Antis and Coad are further reflected in the interlinked, synchronized "views" disclosed in both references. The POSITA in software development systems would have been motivated to supplement Antis's database analysis system with Coad's teaching of a Quality Assurance (QA), because this would allow for easier source code debugging and a more accurate code view display in Antis by enabling automatic detection of source code errors. (*See* Ex. 1015, ¶¶ 164-167.)

4. Burkwald

"Burkwald" (Ex. 1007), U.S. Patent No. 6,356,285, was filed on December 17, 1997, and issued on March 12, 2002. Burkwald is therefore prior art to the '936 patent under 35 U.S.C. § 102(e). Burkwald was applied against the claims of the '936 patent during prosecution. (*See* Ex. 1002, Office Action dated May

31, 2005, at 5-6.) The Office found that Burkwald disclosed limitations of the claims including "graphical representations of data flows [that] are expandable and collapsible." (*See id.*)

Burkwald is directed to a "software visualization technique [that] allows a software application or portfolio to be visually analyzed." (Burkwald at Abstract.) The software visualization technique of Burkwald allows a user to expand or collapse a displayed graphical representation of source code flow. (*Id.* at col. 14, line 43 to col. 15, line 15, and Figs. 6-8.)

5. Motivation to Combine Burkwald with Antis and Coad

A POSITA would have been motivated to combine Burkwald's teachings with Antis and Coad because all three references are directed to software development tools that provide visual representations of source code, such that the similar purposes and overlapping teachings would have motivated the POSITA to combine the teachings of the references. The POSITA would have been motivated to supplement the development tools of Antis and Coad with Burkwald's teaching of expanding and collapsing graphical representations of source code flows, because this would provide a developer with flexibility to modify the views of Antis and Coad to display information at a higher or lower level of detail. (*See* Ex. 1015, ¶ 218.)

6. Access 97 Visual Basic

Access 97 Visual Basic (Ex. 1009) was published in 1997. Access 97 Visual Basic is therefore prior art to the '936 patent under 35 U.S.C. § 102(b).

Access 97 Visual Basic is an instructional book that includes tutorials for using Microsoft Access 97 Visual Basic software, which is a database management system. A portion of the book entitled "Finding and Fixing Bugs in Your Code" discloses that the Microsoft Access 97 Visual Basic software highlights and displays source code in red text when the system determines there is an error. (Access 97 Visual Basic at 140.)

7. Motivation to Combine Access 97 Visual Basic with Antis and Coad

A POSITA would have been motivated to combine the teachings of Access 97 Visual Basic with Antis and Coad because all three references are directed to software development tools that provide visual representations of source code, such that the similar purposes and overlapping teachings would have motivated the POSITA to combine the teachings of the references. As another example, the POSITA in software development systems would have been motivated to supplement the development tools of Antis and Coad with the teachings of Access 97 Visual Basic regarding changing the appearance of source code determined to have errors, because this would simplify source code debugging in Antis and Coad. (*See* Ex. 1015, ¶ 242.)

8. Eick

"Eick" (Ex. 1008), U.S. Patent No. 5,937,064, was filed on March 3, 1997, and issued on August 10, 1999. Eick is therefore prior art to the '936 patent under 35 U.S.C. § 102(b).

Eick is directed to a system for "visual analysis of a database." (Eick at col. 1, lines 6-7.) The disclosed

system allows a variety of remote clients to access information from a database for the purpose of allowing the clients to perform interactive visualization of the database. (*Id. at* col. 2, lines 9-15.) Eick discloses that a security layer can be used to manage the connections between the clients and the database, where the security layer utilizes, for example, password protection or data encryption techniques. (*Id.* at col. 4, lines 19-27.)

9. Motivation to Combine Eick with Antis and Coad

A POSITA would have been motivated to combine Eick's teachings with Antis and Coad because all three references are directed to visual representations of source code and data structures, such that the similar purposes and overlapping teachings would have motivated the POSITA to combine the teachings of the references. As another example, the POSITA in software development systems would have been motivated to supplement the development tools of Antis and Coad with Eick's teaching of a security layer for managing secure connections with remote computers, because this would allow the systems of Antis and Coad to be distributed to one or more remote locations without a substantially increased risk that remotely stored data could be compromised. (*See* Ex. 1015, ¶ 224.)

10. Corda

"Corda" (Ex. 1010), U.S. Patent No. 5,758,122, was filed on March 16, 1995, and issued on May 26, 1998. Corda is therefore prior art to the '936 patent under 35 U.S.C. § 102(b).

Corda is directed to an "immersive visual programming environment" that "permits the flow of data objects and the interaction among the data objects to be visually displayed to the user." (Corda at Abstract.) The disclosed programming environment is configured to change an appearance of a displayed data flow if an error is detected in source code: "Semantic error detection is enhanced through observation of the flow of data and the motion of the structure itself. . . . Run-time error messages in the form of sight and sound alerts are provided, with the offending component highlighted in some way." (*Id.* at col. 4, lines 41-48.)

11. Motivation to Combine Corda with Antis and Coad

A POSITA would have been motivated to combine Corda's teachings with Antis and Coad because all three references are directed to providing visual representations of source code, such that the similar purposes and overlapping teachings would have motivated the POSITA to combine the teachings of the references. As another example, the POSITA in software development systems would have been motivated to supplement the development tools of Antis and Coad with the teachings of Corda regarding changing the appearance of displayed flows to indicate source code determined to have errors, because this would simplify source code debugging in Antis and Coad. (*See* Ex. 1015, ¶ 236.)

12. "Building Applications"

"Building Applications" (Ex. 1011) was published in 1996. "Building Applications" is therefore prior art to the '936 patent under 35 U.S.C. § 102(b).

"Building Applications" is an instructional book that includes tutorials for using Microsoft Access 97 software, which is a database management system. A portion of the book entitled "Writing and Editing Code" discloses that the Microsoft Access 97 software includes an "automatic syntax checking" feature configured to automatically correct segments of source code determined to have errors: "Visual Basic checks the syntax of that line and displays a message if it finds an error. To enable automatic syntax checking, select the Auto Syntax Check check box on the Module tab of the Options dialog box (Tools menu)." ("Building Applications" at 54.)

13. Motivation to Combine "Building Applications" with Antis and Coad

A POSITA would have been motivated to combine the teachings of "Building Applications" with Antis and Coad because all three references are directed to software development tools that provide visual representations of source code, such that the similar purposes and overlapping teachings would have motivated the POSITA to combine the teachings of the references. The POSITA in software development systems would have been motivated to supplement the development tools of Antis and Coad with the teachings of "Building Applications" regarding automatically correcting segments of source code determined to have errors, because this would simplify the debugging of source code in Antis and Coad. (*See* Ex. 1015, ¶ 230.)

14. Oracle Primer

Oracle Primer (Ex. 1012) was published in 1999. Oracle Primer is therefore prior art to the '936 patent under 35 U.S.C. § 102(b).

Oracle Primer is an instructional book that includes information on the Oracle database system. Oracle Primer also includes information on the JDBC (Java Database Connectivity) application programming interface (API), which enables database access in the Java programming language. (Ex. 1012 at 187-189 and 194.)

15. Oracle8 Primer

Oracle8 Primer (Ex. 1013) was published in 1999. (*See* Ex. 1017, Declaration of Morris M. Jackson.) Oracle8 Primer is therefore prior art to the '936 patent under 35 U.S.C. § 102(b).

Oracle8 Primer is an instructional book that includes information on the Oracle database system. Oracle8 Primer discloses that SQL can be embedded in the Java, C, and C++ programming languages, thus allowing each of these programming languages to access data in a database. (Ex. 1013, Oracle8 Primer at 93, 95, 103, 108, 118, 277, 280, 281, 294, and 301.)

16. Motivation to Combine Oracle Primer and Oracle8 Primer with Coad

A POSITA would have been motivated to combine Coad's teachings with Oracle Primer and Oracle8 Primer because all three references are directed to computer programming, generally, and to the Java and C++ programming languages, specifically, such that the similar purposes and overlapping teachings would have motivated the POSITA to combine the

teachings of the references. Further, the POSITA would have been motivated to supplement Coad with the disclosure in the Oracle Primer and the Oracle8 Primer regarding using the Java and C++ programming languages to access data in a database, because this would enhance the utility of these programming languages by allowing them to retrieve, insert, delete, or modify data of a database. (*See* Ex. 1015, ¶ 115.)

B. Claim 1 Is Anticipated by Coad (U.S. Patent No. 6,851,107)

Claim 1 is anticipated by Coad as indicated below.

1. An integrated development environment, comprising:
Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 82-84.)To the extent the preamble is considered a limitation to the claim, Coad discloses an integrated software development environment at Abstract: "Methods and systems consistent with the present invention provide an improved software development tool which allows a developer to simultaneously view a graphical and a textual display of source code. The graphical and textual views are synchronized so that a modification in one view is automatically reflected in the other view." **Additionally, in documents filed in the related litigation, Complementsoft has conceded that this claim element is disclosed in Coad. (Ex. 1016 at 18.)**
[1.1] a document manager for retrieving source code programmed using one of a plurality of types of data

manipulation languages;
Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 85-87.) At col. 15, lines 60-64, Coad discloses a document manager for retrieving source code from a selected file: "As previously stated, the project comprises a plurality of files. The developer either uses the software development tool to open a file which contains existing source code, or to create a file in which the source code will be developed." **At Abstract, Coad further discloses that the retrieved source code may be programmed using one of a plurality of types of programming languages:** "In addition, the software development tool is designed for use with more than one programming language." **As stated above in Section IV.C, a data manipulation language is "*a programming language used to access data in a database, such as to retrieve, insert, delete, or modify data in the database.*" Coad discloses use of the software development tool with the C++ and Java programming languages. (*See, e.g.*, Coad at col. 16, lines 1-4.) Because both of these programming languages had functions that allowed them to access data in a database (*see* Ex. 1015, ¶¶ 48-54), the C++ and Java programming languages of Coad disclose the recited "data manipulation languages" of claim 1.**
[1.2] an editor for displaying the retrieved source

code and providing a means for a user to edit the retrieved source code;
Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 88-89.) At col. 4, lines 54-60, Coad discloses an incremental code editor (ICE) for displaying and editing retrieved source code: "Although modifications made on the displays 204 and 206 may appear to modify the displays 204 and 206, in actuality all modifications are made directly to the source code 202 via an incremental code editor (ICE) 208, and the TMM 200 is used to generate the modifications in both the graphical 204 and the textual 206 views from the modifications to the source code 202." **Additionally, in documents filed in the related litigation, Complementsoft has conceded that this claim element is disclosed in Coad. (Ex. 1016 at 18.)**
[1.3.1] a parser layer which detects the one of the plurality of types of data manipulation languages in which the retrieved source code is programmed and
Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 90-91.) At col. 2, lines 57-58 and col. 5, lines 50-55, Coad discloses a parser layer configured to detect the language of the retrieved source code: "The method comprises the steps of determining the language of the source code of the chosen file . . . ." (Coad at col. 2, lines 57-58.) "The core 700 includes a parser 706 and an ICE 208. The parser 706 converts the source code into the language-neutral representation in the TMM, and

the ICE 208 converts the text from the displays into source code." (*Id.* at col. 5, lines 50-55.)
[1.3.2] [the parser layer] which activates rules and logic applicable to the detected one of the plurality of types of data manipulation languages; and
Coad discloses this claim element. (*See* Ex. 1015, ¶ 92.) Coad discloses that the parser layer is configured to apply rules and logic applicable to the detected language. *See* Coad at col. 5, lines 50-55, reproduced above for element 1.3.1. *See also* col. 16, lines 4-16: "The software development tool then obtains a template for the current programming language, *i.e.*, a collection of generalized definitions for the particular language that can be used to build the data structure (step 904). For example, the definition of a new Java class contains a default name, *e.g.*, 'Class1,' and the default code, 'public class Class1 { }.' Such templates are well known in the art. For example, the 'Microsoft Foundation Class Library' and the 'Microsoft Word Template For Business Use Case Modeling' are examples of standard template libraries from which programmers can choose individual template classes. The software development tool uses the template to parse the source code (step 906), and create the data structure (step 908)." ***See also* Tables 1-17 at col. 6, line 30 to col. 14, line 63, disclosing rules applied by the parser layer.**
[1.4] a visualizer dynamically linked to the editor for displaying graphical representations of flows within the retrieved source code using the rules and logic

applicable to the detected one of the plurality of types of data manipulation languages and activated by the parser,
Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 93-107.) Coad discloses a viewer (*i.e.*, visualizer) for displaying graphical representations of flows in source code at col. 16, line 57 to col. 17, line 47: "The software development tool is collectively broken into three views of the application: the static view, the dynamic view, and the functional view. The static view is modeled using the use-case and class diagrams. A use case diagram 1200, depicted in FIG. 12, shows the relationship among actors 1202 and use cases 1204 within the system 1206. A class diagram 1300, depicted in FIG. 13 with its associated source code 1302, on the other hand, includes classes 1304, interfaces, packages and their relationships connected as a graph to each other and to their contents. The dynamic view is modeled using the sequence, collaboration and statechart diagrams. As depicted in FIG. 14, a sequence diagram 1400 represents an interaction, which is a set of messages 1402 exchanged among objects 1404 within a collaboration to effect a desired operation or result. In a sequence diagram 1400, the vertical dimension represents time and the horizontal dimension represents different objects. A collaboration diagram 1500, depicted in FIG. 15, is also an interaction with messages 1502 exchanged among objects 1504, but it is also a collaboration, which is a set of objects 1504 related in a particular context. Contrary to sequence diagrams

1400 (FIG. 14), which emphasize the time ordering of messages along the vertical axis, collaboration diagrams 1500 (FIG. 15) emphasize the structural organization of objects.

A statechart diagram 1600 is depicted in FIG. 16. The statechart diagram 1600 includes the sequences of states 1602 that an object or interaction goes through during its life in response to stimuli, together with its responses and actions. It uses a graphic notation that shows states of an object, the events that cause a transition from one state to another, and the actions that result from the transition.

The functional view can be represented by activity diagrams 1700 and more traditional descriptive narratives such as pseudocode and minispecifications. An activity diagram 1700 is depicted in FIG. 17, and is a special case of a state diagram where most, if not all, of the states are action states 1702 and where most, if not all, of the transitions are triggered by completion of the actions in the source states. Activity diagrams 1700 are used in situations where all or most of the events represent the completion of internally generated actions.

There is also a fourth view mingled with the static view called the architectural view. This view is modeled using package, component and deployment diagrams. Package diagrams show packages of classes and the dependencies among them. Component diagrams 1800, depicted in FIG. 18, are graphical representations of a system or its component parts. Component diagrams 1800 show

the dependencies among software components, including source code components, binary code components and executable components. As depicted in FIG. 19, Deployment diagrams 1900 are used to show the distribution strategy for a distributed object system. Deployment diagrams 1900 show the configuration of run-time processing elements and the software components, processes and objects that live on them." **Figs. 11 – 17 of Coad depict aspects of the viewer for displaying graphical representations of flows in source code. For example, Fig. 16 is an example of a graphical representation of flow disclosed in Coad, where a statechart diagram 1600 depicts the sequence of states 1602 that an object or interaction goes through during its life in response to stimuli.**
[1.5] wherein the editor, parser layer and visualizer cooperate such that edits made to the source code using the editor are automatically reflected in the graphical representations of flows displayed by the visualizer and edits made to the graphical representations of flows in the visualizer are automatically reflected in the source code displayed by the editor.
Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 108-110.) Coad discloses that graphical and textual views are synchronized such that modifications made to one view are automatically reflected in the other view, as disclosed at col. 4, line 61 to col. 5, line 3: "The improved software development tool provides simultaneous round-trip engineering, *i.e.*, the

> graphical representation 204 is synchronized with the textual representation 206. Thus, if a change is made to the source code 202 via the graphical representation 204, the textual representation 206 is updated automatically. Similarly, if a change is made to the source code 202 via the textual representation 206, the graphical representation 204 is updated to remain synchronized. There is no repository, no batch code generation, and no risk of losing code."

Complementsoft's arguments made with respect to Coad during prosecution are inconsistent with the claim language and lack technical merit. In particular, the argument that equated "data manipulation languages" with "unstructured programming languages" is technically untrue because a person of ordinary skill in the art would understand that most data manipulation languages are in fact structured programming languages. (*See* Ex. 1015, ¶¶ 24-60.)

Further, the scope of claim 1, covering both program flows *and* data flows, is inconsistent with the argument made by Complementsoft when distinguishing claim 1 from Coad. This is because the distinction argued by Complementsoft (*i.e.*, that displaying flow in unstructured programming languages is distinguishable from displaying flow in structured programming languages) applies, if at all, only to displaying *program flows*. This distinction does not apply to data flows. (*Id.* at ¶¶ 61-68.)

For these reasons, and in light of the fact that Coad discloses a "data manipulation language" (*see* Ex. 1015, ¶¶ 48-54), Petitioner believes that the

Office's rejection of claim 1 over Coad was proper and therefore proposes a similar ground of unpatentability here.

C. Claim 1 Is Obvious Over Coad in View of Oracle Primer and Oracle8 Primer

As discussed above, Coad discloses all limitations of claim 1. To the extent it could be argued that any further disclosure may be required with respect to the "data manipulation language" feature of claim 1, Oracle Primer and Oracle8 Primer provide such further disclosure. As explained above in Section IV.C, a data manipulation language is *a programming language used to access data in a database, such as to retrieve, insert, delete, or modify data in the database.* The Oracle Primer and Oracle8 Primer references disclose that the programming languages used in the Coad reference (*i.e.*, Java and C++) included functions that allowed the programming languages to access data in a database. Therefore, Oracle Primer and Oracle8 Primer disclose that the Java and C++ programming languages of Coad are "data manipulation languages," as recited in claim 1. (*See* Ex. 1015, ¶¶ 111-114.)

For example, Oracle Primer discloses the JDBC (Java Database Connectivity) application programming interface (API), which enables database access in the Java programming language. (Ex. 1012, Oracle Primer at 187-189 and 194.) The Oracle8 Primer discloses that SQL can be embedded in the Java, C, and C++ programming languages, thus allowing each of these languages to access data

in a database. (Ex. 1013, Oracle8 Primer at 93, 95, 103, 108, 118, 277, 280, 281, 294, and 301.)

Oracle Primer and Oracle8 Primer are combinable with Coad for the reasons stated in Section V.A.16. (*See also* Ex. 1015, ¶ 115.)

D. Claims 1-3 and 5 Are Anticipated by Antis (U.S. Patent No. 5,572,650)

1. Claim 1

Claim 1 is anticipated by Antis as indicated below.

1. An integrated development environment, comprising:
Antis discloses this claim element. (*See* Ex. 1015, ¶¶ 125-127.) To the extent the preamble is considered a limitation to the claim,[3] Antis discloses a relational database analysis system including a plurality of software development features, at col. 2, lines 25-29; col. 3, lines 23-30; and Abstract: "Therefore, there is a long felt need in the art for a method and apparatus for displaying characteristics of a database without semantic information such that explicit and implicit data structures can be readily observed to facilitate *use, development and maintenance of large databases.*" (Antis at col. 2, lines 25-29, emphasis added.) "Referring now to FIG. 1, a block diagram of an example of a relational database analysis system 101 is shown. The system 101 includes terminal 103,

[3] *See* MPEP § 2111.02 Effect of Preamble ("The determination of whether a preamble limits a claim is made on a case-by-case basis in light of the facts in each case").

which provides output to and receives input from a user of the system, processor 113, which performs the actual analysis operations, memory system 115, which contains programs 117 executed by processor 113 and relations 119cI19i, each of which contains a respective set of attributes and tuples." (*Id.* at col. 3, lines 23-30.)

"A method and apparatus for visually displaying structural characteristics of a large database for visualizing areas where new developments may be added, or finding places where old developments may need to be re-structured or replaced." (*Id.* at Abstract.)

[1.1] a document manager for retrieving source code programmed using one of a plurality of types of data manipulation languages;

Antis discloses this claim element. (*See* Ex. 1015, ¶¶ 128-131.) At col. 9, lines 45-46, Antis discloses the use of Cscope, a "well known code browser," to retrieve source code programmed using one of a plurality of types of data manipulation languages.

"[T]he source code references to the relations are extracted from a Cscope database (Cscope is a well known code browser)."

Antis also discloses an "over view" display 201 of the relational database analysis system, which is described at col. 4, lines 2-3, and depicted at Fig. 2:

"Display 201 is an over view, which is the highest view level of the large relational database."

The over view can be used to retrieve source

code via a plurality of different methods. For example, as illustrated in Fig. 2, the display 201 of the over view includes activity buttons near the top of the GUI window. The activity buttons allow different views of the relational database analysis system to be invoked, including a "code view." From the code view, the aforementioned Cscope code browser is used to retrieve source code:

"Across the top of display 201 are a number of activity buttons which may be activated by the cursor to provide additional information about the selected relation, in this case relation RLls_lnk. The buttons are labeled: Assocs (abbreviation for Associations), Paths, Code, PRL (abbreviation for Population Rule Language, commonly referred to as the Specification View), Layout, Relations List, Domains, Entity, and Distance. *Each of these buttons, except Distance, may be used to open another window on the screen 105 (see FIG. 1) which displays another view of the relational database under analysis.*" (Antis at col. 4, line 60 to col. 5, line 3, emphasis added.)

Source code can also be retrieved from the over view by selecting one of the relations displayed in the over view display 201. As disclosed at col. 7, lines 32-35, based on the relation selected in the over view, source code that accesses the selected relation is retrieved:

"The code view displays the application source code of the RDBMS that uses the currently selected relation, in this case relation RLls_lnk."

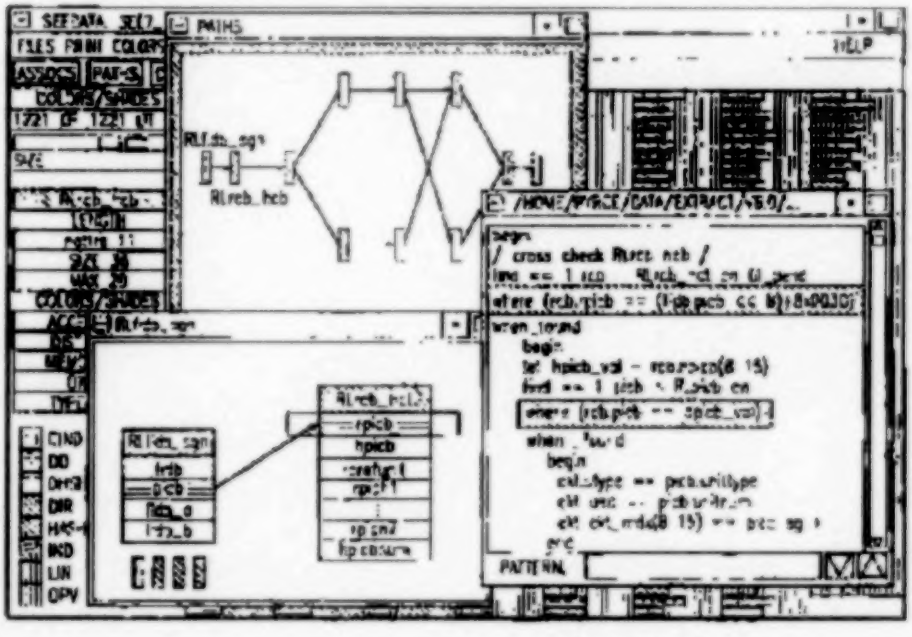
Additionally, in documents filed in the related litigation, Complementsoft has conceded that

this claim element is disclosed in Antis. (Ex. 1016 at 3.)

[1.2] an editor for displaying the retrieved source code and providing a means for a user to edit the retrieved source code;

Antis discloses this claim element. (*See* Ex. 1015, ¶¶ 132-134.) Antis discloses an editor for displaying and editing retrieved source code in an "expanded code view," as illustrated at the bottom right hand corner of Fig. 12 and as described at col. 8, lines 63-66:

FIG. 12



"FIG. 12 shows an over view, a relations view, a path view and an expanded code view all displayed in partially overlapping windows on a display screen."

The expanded code view is used to edit the retrieved source code. Antis discloses that changes made in any of the disclosed views (*e.g.*, edits to the source code made in the expanded code view or a definition of a new

object) cause corresponding changes in the other views, at col. 9, lines 17-28:

"The overall view is purely structural without semantic details to clutter it. If semantic details are desired, they are available in views called the specification view and the code view. If further structural details of a particular relation are desired, further details are available in the associations view, the path view, the code view and the domain view. *The linking of the views makes them interactive with each other such that changes to one view causes corresponding changes in the other actively linked views. This not only speeds up analysis, but also gives the user a chance to analyze the effects of any changes, e.g. definition of a new object, on the relational database system.*" (Emphasis added.)

[1.3.1] a parser layer which detects the one of the plurality of types of data manipulation languages in which the retrieved source code is programmed and

Antis discloses this claim element. (*See* Ex. 1015, ¶¶ 135-137.) At col. 9, lines 35-41, Antis discloses a parser layer based on the YACC computer program. The parser layer disclosed in Antis can be used for "any type of database" and is thus used to detect any of a plurality of data manipulation languages having "a YACC grammar":

"*For any type of database*, the associations could be extracted from the database query language with an A * version (A* is a combination of pattern matching that is based on AWK, and parsing that is based on YACC which allows AWK style pattern matching to be applied to grammar productions of *a language*

specified with a YACC grammar) built for that language grammar." (Emphasis added.) ***See also id.* at col. 5, lines 4-8, disclosing applicability of the relational database analysis system to multiple different data manipulation languages:** "Referring now to FIG. 3, a display 301 is shown on display screen 105 that is called the specification view. In the specification view the actual specification(s) of the database *in the database description language or languages that are understood by the RDBMS may be directly viewed.*" (Emphasis added.) **Additionally, in documents filed in the related litigation, Complementsoft has conceded that this claim element is disclosed in Antis. (Ex. 1016 at 3.)**
[1.3.2] [the parser layer] which activates rules and logic applicable to the detected one of the plurality of types of data manipulation languages; and
Antis discloses this claim element. (*See* Ex. 1015, ¶ 138.) Antis discloses that the parser layer based on the YACC computer program activates rules and logic for a detected one of the plurality of types of languages. *See* Antis at col. 9, lines 35-41, reproduced above for element 1.3.1. Additionally, in documents filed in the related litigation, Complementsoft has conceded that this claim element is disclosed in Antis. (Ex. 1016 at 3.)
[1.4] a visualizer dynamically linked to the editor for displaying graphical representations of flows within

the retrieved source code using the rules and logic applicable to the detected one of the plurality of types of data manipulation languages and activated by the parser,
Antis discloses this claim element. (*See* Ex. 1015, ¶¶ 139-146.) Antis discloses a visualizer for displaying graphical representations of flows within the retrieved source code in a "code view," as illustrated at Fig. 8 and as described at col. 7, lines 31-45. Specifically, the code view provides a graphical display of flow in the retrieved source code by providing a visual representation of the data within the relational database (*i.e.*, the relations) that are accessed by the retrieved source code. The code view also depicts a graphical display of flow by providing a visual representation of which pieces of source code are executed for a selected relation, where the relation is selected via the over view display described above: "Referring now to FIG. 8, a display 801 is shown on display screen 105 that is called the code view. The code view displays the application source code of the RDBMS that uses the currently selected relation, in this case relation RLls_lnk. *The application source code may be represented as a horizontal tree representing the code hierarchy. The subset or subsets of that horizontal code tree that use the currently selected relation are displayed in the code view. The code view may be used by the user to find the 40 application source code which references a relation and also to find where in the application source code the relation is accessed. The code view is linked to the over view such that all relations accessed from a unit*

of application code may be easily identified. Each unit of application code is represented 45 by a box in the code view." (Emphasis added.)

The code view of Antis further discloses a graphical representation of flow within the retrieved code at col. 7, lines 51-53. This portion discloses that clicking on a code box (*i.e.*, an icon associated with a particular piece of source code) produces a visual representation of the data in the relational database (*i.e.*, the relations) that are accessed by the particular piece of source code:

"Clicking the mouse button on a code box highlights in the over view all relations that use the corresponding unit of code."

Antis further discloses a graphical representation of flows within the retrieved source code at col. 7, lines 57-61. This portion discloses that a parallel flow of execution is displayed in the code view if the retrieved source code is executed on multiple processors (*i.e.*, the code view display will be changed depending on the number of processors executing the retrieved source code, thus providing a graphical display of the flow of execution for the code):

"For the purposes of the code view, application code executing on multiple separate processors in a distributed system is considered to be multiple separate applications, each of which may be examined as a separate code sub-tree."

[1.5] wherein the editor, parser layer and visualizer cooperate such that edits made to the source code

using the editor are automatically reflected in the graphical representations of flows displayed by the visualizer and edits made to the graphical representations of flows in the visualizer are automatically reflected in the source code displayed by the editor.
Antis discloses this claim element. (*See* Ex. 1015, ¶¶ 147-149.) Antis discloses that the various views (*i.e.*, the over view, code view, etc.) of the of the relational database analysis system are linked, such that changes made in one view are automatically made in the other linked views, at col. 9, lines 17-28: "The overall view is purely structural without semantic details to clutter it. If semantic details are desired, they are available in views called the specification view and the code view. If further structural details of a particular relation are desired, further details are available in the associations view, the path view, *the code view* and the domain view. *The linking of the views makes them interactive with each other such that changes to one view causes corresponding changes in the other actively linked views.* This not only speeds up analysis, but also gives the user a chance to analyze the effects of any changes, *e.g.* definition of a new object, on the relational database system." (Emphasis added.)

2. Claim 2

Claim 2 is anticipated by Antis as indicated below.

2. The integrated development environment as recited in claim 1, wherein the graphical representations of flows depict data flows.

Antis discloses this claim element. (*See* Ex. 1015, ¶¶ 150-153.) The code view of Antis discloses graphical representations of data flows within the retrieved source code, as described at col. 7, lines 31-45 (reproduced above for element 1.4 of claim 1). Specifically, the code view provides a visualization of data flows in the retrieved source by providing a visual representation of which pieces of source code access which data in the relational database.

3. Claim 3

Claim 3 is anticipated by Antis as indicated below.

3. The integrated development environment as recited in claim 1, wherein the graphical representations of flows depict program flows.

Antis discloses this claim element. (*See* Ex. 1015, ¶¶ 154-157.) The code view of Antis discloses graphical representations of program flows within the retrieved source code, as described at col. 7, lines 31-45 (reproduced above for element 1.4 of claim 1). Specifically, the code view provides a visualization of program flows in the retrieved source by providing a visual representation of which pieces of source code are executed for a selected relation of the database. Antis further discloses a graphical representation of program flows at col. 7, lines 57-61. This portion discloses that a parallel flow of execution (*i.e.*, program flows) is displayed in the code view if the retrieved source code is executed on multiple processors:

"For the purposes of the code view, application code executing on multiple separate processors in a distributed system is considered to be multiple separate applications, each of which may be examined as a separate code sub-tree." (Antis at col. 7, lines 57-61.)

4. Claim 5

Claim 5 is anticipated by Antis as indicated below.

5. The integrated development environment as recited in claim 1, wherein the document manager retrieves all files related to the source code to be edited.

Antis discloses this claim element. (*See* Ex. 1015, ¶¶ 158-161.) At col. 7, lines 31-35, Antis discloses that the code view is used to retrieve source code related to a relation that is selected in the over view. Further, as explained above with reference to claim 1, the expanded code view of Antis depicted in Fig. 12 allows the retrieved code to be edited:

"Referring now to FIG. 8, a display 801 is shown on display screen 105 that is called the code view. The code view displays the application source code of the RDBMS that uses the currently selected relation, in this case relation RLls_lnk."

Further, at col. 7, lines 39-44, Antis discloses that all pieces of source code related to the selected relation are retrieved:

"The code view may be used by the user to find the 40 application source code which references a relation and also to find where in the application source code

the relation is accessed. The code view is linked to the over view such that all relations accessed from a unit of application code may be easily identified." **Additionally, in documents filed in the related litigation, Complementsoft has conceded that this claim element is disclosed in Antis. (Ex. 1016 at 5.)**

E. Claims 1-3, 5, 6, 8, 10-12, 15, and 16 Are Obvious Over Antis in View of Coad

1. Claim 1

As discussed above, Antis discloses all limitations of claim 1. To the extent it could be argued that any further disclosure may be required with respect to limitations 1.2, 1.4, and 1.5, Coad provides such further disclosure as follows.

[1.2] an editor for displaying the retrieved source code and providing a means for a user to edit the retrieved source code;
Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 88-89.) Coad discloses an incremental code editor (ICE) for displaying and editing retrieved source code. *See* Coad at col. 4, lines 54-60, reproduced above for element 1.2 of claim 1. Additionally, in documents filed in the related litigation, Complementsoft has conceded that this claim element is disclosed in Coad. (Ex. 1016 at 18.)
[1.4] a visualizer dynamically linked to the editor for displaying graphical representations of flows within the retrieved source code using the rules and logic applicable to the detected one of the plurality of types

of data manipulation languages and activated by the parser,
Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 93-107.) Coad discloses a viewer (*i.e.*, visualizer) for displaying graphical representations of flows in source code at col. 16, line 57 to col. 17, line 47, reproduced above for element 1.4 of claim 1. Further, Figs. 11 – 17 of Coad depict aspects of the viewer for displaying graphical representations of flows in source code. For example, Fig. 16 is an example of a graphical representation of flow disclosed in Coad, where a statechart diagram 1600 depicts the sequence of states 1602 that an object or interaction goes through during its life in response to stimuli.
[1.5] wherein the editor, parser layer and visualizer cooperate such that edits made to the source code using the editor are automatically reflected in the graphical representations of flows displayed by the visualizer and edits made to the graphical representations of flows in the visualizer are automatically reflected in the source code displayed by the editor.
Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 108-110.) Coad discloses a software development tool that allows a developer to simultaneously view a graphical representation and a text representation of source code. (*See, e.g.* Coad at Abstract and Fig. 2.) The graphical and textual views are synchronized such that modifications made to one view are automatically reflected in the other view, as

disclosed at col. 4, line 61 to col. 5, line 3 (reproduced above for element 1.5 of claim 1).

Coad is combinable with Antis for the reasons stated in Section V.A.3. (*See also* Ex. 1015, ¶¶ 164-167.) The proposed ground of unpatentability based on Antis and Coad is included because Coad provides additional details on the visualization of flow that are not provided in Antis. The proposed ground of unpatentability is further included because the Office previously found that Coad disclosed these claim limitations, and this finding was not disputed by Complementsoft. (Ex. 1002, Office Action dated May 31, 2005, at 3, and Ex. 1003, Amendment dated August 4, 2005, at 9-12.)

2. Claim 2

As discussed above, Antis discloses all limitations of claim 2. To the extent it could be argued that any further disclosure may be required with respect to the limitations of claim 2, Coad provides such further disclosure as follows.

2. The integrated development environment as recited in claim 1, wherein the graphical representations of flows depict data flows.
Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 168-171.) Coad discloses a visualizer for displaying graphical representations of data flows at col. 16, line 57 to col. 17, line 47, reproduced above for element 1.4 of claim 1.

Coad is combinable with Antis for the reasons stated in Section V.A.3. (*See also* Ex. 1015, ¶¶ 164-

167.) The proposed ground of unpatentability based on Antis and Coad is included because Coad provides additional details on the visualization of data flow that are not provided in Antis. The proposed ground of unpatentability is further included because the Office previously found that Coad disclosed this claim limitation, and this finding was not disputed by Complementsoft. (Ex. 1002, Office Action dated May 31, 2005, at 4, and Ex. 1003, Amendment dated August 4, 2005, at 9-12.)

3. Claim 3

As discussed above, Antis discloses all limitations of claim 3. To the extent it could be argued that any further disclosure may be required with respect to the limitations of claim 3, Coad provides such further disclosure as follows.

3. The integrated development environment as recited in claim 1, wherein the graphical representations of flows depict program flows.
Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 172-175.) Coad discloses a visualizer for displaying graphical representations of program flows at col. 16, line 57 to col. 17, line 47, reproduced above for element 1.4 of claim 1.

Coad is combinable with Antis for the reasons stated in Section V.A.3. (*See also* Ex. 1015, ¶¶ 164-167.) The proposed ground of unpatentability based on Antis and Coad is included because Coad provides additional details on the visualization of program flow that are not provided in Antis. The proposed ground of unpatentability is further included because the Office previously found that Coad disclosed this

claim limitation, and this finding was not disputed by Complementsoft. (Ex. 1002, Office Action dated May 31, 2005, at 4, and Ex. 1003, Amendment dated August 4, 2005, at 9-12.)

4. Claim 5

Claim 5 depends from claim 1, which, as discussed above, is obvious over Antis in view of Coad. As explained in Section V.D.4, Antis discloses the additional limitations in claim 5, which accordingly is obvious over Antis in view of Coad. (*See* Ex. 1015, ¶ 176.)

5. Claim 6

Claim 6 depends from claim 1, which, as discussed above, is (1) anticipated by Antis and (2) obvious over Antis in view of Coad. Coad discloses the additional limitations in claim 6, which accordingly is obvious over Antis in view of Coad.

6. The integrated development environment as recited in claim 1, wherein the document manager comprises a site manager and a connectivity layer for retrieving source code from one or more remote computers.
Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 177-180.) At col. 5, lines 31-49, Coad discloses that an incremental code editor (ICE) (*see, e.g.*, Fig. 2 and col. 4, line 58) includes site manager and connectivity layer functionality for retrieving source code from remote computers over the Internet: "FIG. 6 depicts a data processing system 600 suitable for practicing methods and systems consistent with the present invention. Data processing system 600

comprises a memory 602, a secondary storage device 604, an I/O device 606, and a processor 608. Memory 602 includes the improved software development tool 610. The software development tool 610 is used to develop a software project 612, and create the TMM 200 in the memory 602. The project 612 is stored in the secondary storage device 604 of the data processing system 600. One skilled in the art will recognize that data processing system 600 may contain additional or different components.

Although aspects of the present invention are described as being stored in memory, one skilled in the art will appreciate that these aspects can also be stored on or read from other types of computer-readable media, such as secondary storage devices, like hard disks, floppy disks or CD-ROM; a carrier wave from a network, such as Internet; or other forms of RAM or ROM either currently known or later developed."

Additionally, in documents filed in the related litigation, Complementsoft has conceded that this claim element is disclosed in Coad. (Ex. 1016 at 20-21.)

Coad is combinable with Antis for the reasons stated in Section V.A.3. (*See also* Ex. 1015, ¶¶ 164-167.) During prosecution, the Office found that Coad disclosed this claim limitation, and Complementsoft did not dispute this finding. (Ex. 1002, Office Action dated May 31, 2005, at 4, and Ex. 1003, Amendment dated August 4, 2005, at 9-12.)

6. Claim 8

Claim 8 depends from claim 1, which, as discussed above, is (1) anticipated by Antis and (2) obvious over Antis in view of Coad. Coad discloses the additional limitations in claim 8, which accordingly is obvious over Antis in view of Coad.

8. The integrated development environment as recited in claim 1, wherein the editor comprises a template manager for allowing preprogrammed segment of source code to be placed within the source code being edited.
Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 181-185.) At col. 16, lines 4-9, Coad discloses "a collection of generalized definitions for the particular language that can be used to build [a] data structure." The generalized definitions include "default code" that can be placed within source code being edited. "The software development tool then obtains a template for the current programming language, *i.e.*, a collection of generalized definitions for the particular language that can be used to build the data structure (step 904). For example, the definition of a new Java class contains a default name, *e.g.*, 'Class1,' and the default code, 'public class Class1 {}.' Such templates are well known in the art."

Coad is combinable with Antis for the reasons stated in Section V.A.3. (*See also* Ex. 1015, ¶¶ 164-167.)

7. Claim 10

Claim 10 depends from claim 8, which, as discussed above, is obvious over Antis in view of Coad. Coad discloses the additional limitations in claim 10, which accordingly is obvious over Antis in view of Coad.

10. The integrated development environment as recited in claim 8, wherein the template manager is adapted to automatically generate segments of the source code.
Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 186-190.) Coad discloses that the template manager is configured to automatically generate segments of source code. *See* Coad at col. 16, lines 4-9, reproduced above for claim 8.

Coad is combinable with Antis for the reasons stated in Section V.A.3. (*See also* Ex. 1015, ¶¶ 164-167.)

8. Claim 11

Claim 11 depends from claim 1, which, as discussed above, is (1) anticipated by Antis and (2) obvious over Antis in view of Coad. Coad and Antis disclose the additional limitations in claim 11, which accordingly is obvious over Antis in view of Coad.

11. The integrated development environment as recited in claim 1, further comprising a means for allowing the source code to be executed both locally and remotely.
Coad and Antis disclose this claim element. (*See* Ex. 1015, ¶¶ 191-195.) Coad discloses that

the ICE editor includes functionality for allowing retrieved source to be executed locally (*i.e.*, from a local memory) or remotely (*i.e.*, via the Internet). *See* Coad at col. 5, lines 31-49, reproduced above for claim 6. Further, at col. 7, lines 55-61, Antis discloses a distributed system with remote execution of source code:

"If there are multiple applications that use the same relation, the result will be multiple code sub-trees in the code view, as sub-trees 805 and 809. For the purposes of the code view, application code executing on multiple separate processors in a distributed system is considered to be multiple separate applications, each of which may be examined as a separate code sub-tree."

Coad is combinable with Antis for the reasons stated in Section V.A.3. (*See also* Ex. 1015, ¶¶ 164-167.) During prosecution, the Office found that Coad disclosed this claim limitation. (Ex. 1002, Office Action dated May 31, 2005, at 4.)

9. Claim 12

Claim 12 depends from claim 11, which, as discussed above, is obvious over Antis in view of Coad. Coad discloses the additional limitations in claim 12, which accordingly is obvious over Antis in view of Coad.

12. The integrated development environment as recited in claim 11, wherein the parser layer further examines error log files generated by the means for allowing the source code to be executed to determine segments of the source code determined to include

errors.
Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 196-201.) At col. 9, line 66 to col. 10, line 5, Coad discloses a quality assurance (QA) module for detecting segments of source code containing errors. If the QA module determines that the source code "does not conform, an error message is provided to the developer" (*see* Coad at col. 14, lines 64-65): "The QA module also provides audits, *i.e.*, the module checks for conformance to predefined or user-defined styles. The types of audits provided by the module include coding style, critical errors, declaration style, documentation, naming style, performance, possible errors and superfluous content. Examples of these audits with their respective definitions are identified in Tables 10-17 below." ***See also* Coad at col. 10, line 8 to col. 14, line 63, disclosing Tables 10-17, including examples of coding style audits, critical error audits, and declaration style audits, among others.**

Coad is combinable with Antis for the reasons stated in Section V.A.3. (*See also* Ex. 1015, ¶¶ 164-167.) During prosecution, the Office found that Coad disclosed this claim limitation. (Ex. 1002, Office Action dated May 31, 2005, at 4.)

10. Claim 15

Claim 15 depends from claim 12, which, as discussed above, is obvious over Antis in view of Coad. Coad discloses the additional limitations in

claim 15, which accordingly is obvious over Antis in view of Coad.

15. The integrated development environment as recited in claim 12, further comprising a message manager cooperating with the parser layer for displaying debugging hints as a function of the source code segments determined to have errors.
Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 202-208.) Coad discloses a quality assurance (QA) module for determining segments of source code including errors. *See* Coad at col. 9, line 66 to col. 10, line 5, reproduced above for claim 12. At col. 14, line 64 to col. 15, line 16, Coad further discloses that if errors in the source code are detected by the QA module, an error message is provided to the developer: "If the QA module determines that the source code does not conform, an error message is provided to the developer. For example, as depicted in FIG. 8A, the software development tool checks for a variety of coding styles 800. If the software development tool were to check for 'Access Of Static Members Through Objects' 802, it would verify whether static members are referenced through class names rather than through objects 804. Further, as depicted in FIG. 8B, if the software development tool were to check for 'Complex Assignment' 806, the software development tool would check for the occurrence of multiple assignments and assignments to variables within the same expression to avoid complex assignments since these decrease program readability 808. An example of source code having a complex assignment 810 and

source code having a non-complex assignment 812 are depicted in FIGS. 8B and 8C, respectively. The QA module of the software development tool scans the source code for other syntax errors well known in the art, as described above, and provides an error message if any such errors are detected."

At Fig. 8B, Coad discloses a message manager for displaying debugging hints for a variety of errors. As illustrated in Fig. 8B, the errors include, for example, "Complex Assignment" and "Don't Use the Negation Operator Frequently" errors. An example debugging hint is illustrated in Fig. 8B ("Tip: Break statement into several ones.").

Coad is combinable with Antis for the reasons stated in Section V.A.3. (*See also* Ex. 1015, ¶¶ 164-167.) During prosecution, the Office found that Coad disclosed this claim limitation, and this finding was not disputed by Complementsoft. (Ex. 1002, Office Action dated May 31, 2005, at 4, and Ex. 1003, Amendment dated August 4, 2005, at 9-12.)

11. Claim 16

Claim 16 depends from claim 15, which, as discussed above, is obvious over Antis in view of Coad. Coad discloses the additional limitations in claim 16, which accordingly is obvious over Antis in view of Coad.

16. The integrated development environment as recited in claim 15, wherein the message manager allows a user to edit and maintain debugging hints for a variety of different errors.

Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 209-213.) At Fig. 8B, Coad discloses a message manager that maintains debugging tips for a variety of errors. As illustrated in Fig. 8B, the errors include, for example, "Complex Assignment" and "Don't Use the Negation Operator Frequently" errors. Corresponding debugging tips are maintained for each of the errors.

Coad is combinable with Antis for the reasons stated in Section V.A.3. (*See also* Ex. 1015, ¶¶ 164-167.) During prosecution, the Office found that Coad disclosed this claim limitation, and this finding was not disputed by Complementsoft. (Ex. 1002, Office Action dated May 31, 2005, at 4, and Ex. 1003, Amendment dated August 4, 2005, at 9-12.)

F. Claim 4 Is Obvious Over Antis in View of Coad and Burkwald (U.S. Patent No. 6,356,285)

Claim 4 depends from claim 1, which, as discussed above, is obvious over Antis in view of Coad. Burkwald discloses the additional limitations in claim 4, which accordingly is obvious over Antis in view of Coad and Burkwald.

4. The integrated development environment as recited in claim 1, wherein the graphical representations of data flows are expandable and collapsible.

Burkwald discloses this claim element. (*See* Ex. 1015, ¶¶ 214-217.) At col. 14, line 43 to col. 15, line 4, Burkwald discloses a software

visualization technique that allows a user to expand or collapse a displayed graphical representation of source code:

"In FIGS. 6-8, each bar 612 of the bar chart 610 represents one of the subsystems of the large systems 110 or one of the small systems 120 of the exemplary legacy software application. As shown in the bar chart 610 of FIGS. 6-8, the height of each bar 612 is proportional to the numbers of the programs 114 in each of the subsystems 116, sorted in decreasing order. For each program 114 in the legacy software application, the collapsed values list 620 shows both the McCabe complexity metric 622 and the McClure complexity metric 624, the total number of lines 626, and the total number of affected lines 628. It should be appreciated that, while FIGS. 6-8 show the collapsed values list 620 using the above-outlined metrics and statistical values for the software subsystem, the collapsed values list 620 can be used to show collapsed values for any metric and/or any statistic for which values have been obtained. That is, the analyst can interactively change the displayed metrics and statistics to any metric and/or statistic for which the analyst has generated data for the software application being displayed.

By manipulating the zoom bar 629 of the values list 620, the user can expand or collapse the values list 620 shown in FIG. 6 to control the height consumed by each line of data. In the views 600-800 shown in FIGS. 6-8, the values list 620 has been collapsed so that each line of data is shown as a one-pixel high row. Furthermore, the rows have been sorted in decreasing order according to the number of affected lines 628. The length of each row encodes the value

of the corresponding metric."

Burkwald is combinable with Antis and Coad for the reasons stated in Section V.A.5. (*See also* Ex. 1015, ¶ 218.) During prosecution, the Office found that Burkwald disclosed this claim limitation, and this finding was not disputed by Complementsoft. (Ex. 1002, Office Action dated May 31, 2005, at 4-6, and Ex. 1003, Amendment dated August 4, 2005, at 9-12.)

G. Claim 7 Is Obvious Over Antis in view of Coad and Eick (U.S. Patent No. 5,937,064)

Claim 7 depends from claim 6, which, as discussed above, is obvious over Antis in view of Coad. Eick discloses the additional limitations in claim 7, which accordingly is obvious over Antis in view of Coad and Eick.

7. The integrated development environment as recited in claim 6, wherein the document manager comprises a security layer for managing secure connections with the one or more remote computers.

Eick discloses this claim element. (*See* Ex. 1015, ¶¶ 219-223.) Eick discloses a "system and method for visual analysis of a database." (Eick at Abstract.) The disclosed system and method allow a variety of remote clients to access information from a database for the purpose of allowing the clients to perform interactive visualization of the database. At col. 4, lines 19-27, Eick discloses a security layer for managing secure connections with the remote clients:

"Two approaches may be used to permit only authorized access to the database or authorization to change the database. One approach from the web server side would be to pass data or access to the data base to the client upon showing the appropriate password. The other would be to have the CGI 22 scripts paths translated to pass encrypted data and make the data available and then pass the data to the applet which would then decide what data to show the client in view of establishing the appropriate security."

***See also id.* at claims 7 and 8:**

"7. The method of claim 6 in which said live document program includes a security subprogram which prevents display of at least one selected part of the data unless preselected security requirements are satisfied at the at least one computer terminal."

"8. The method of claim 7 including the step of selectively decoding encrypted data only if the preselected security requirements are satisfied."

Additionally, in documents filed in the related litigation, Complementsoft has conceded that this claim element is disclosed in Eick. (Ex. 1016 at 6.)

Eick is combinable with Antis and Coad for the reasons stated in Section V.A.9. (*See also* Ex. 1015, ¶ 224.)

H. Claim 9 Is Obvious Over Antis in view of Coad and "Building Applications"

Claim 9 depends from claim 8, which, as discussed above, is obvious over Antis in view of Coad.

"Building Applications" discloses the additional limitations in claim 9, which accordingly is obvious over Antis in view of Coad and "Building Applications."

9. The integrated development environment as recited in claim 8, wherein the template manager is adapted to automatically correct segments of the source code.
"Building Applications" discloses this claim element. (*See* Ex. 1015, ¶¶ 225-229.) At page 54, "Building Applications" discloses that "automatic syntax checking" may be enabled, which allows for syntax errors to be detected and corresponding error messages to be displayed: "Automatic syntax checking – As you move the insertion point off a line, Visual Basic checks the syntax of that line and displays a message if it finds an error. To enable automatic syntax checking, select the Auto Syntax Check check box on the Module tab of the Options dialog box (Tools menu)." **Further, at pages 52-53, "automatic statement building" is used to assist the user in writing source code by providing pre-written segments of source code that can be selected by the user. Thus, the automatic syntax checking informs the user of errors in the source code, and the automatic statement building provides segments of source code to the user, such that the errors can be automatically corrected:** "**Automatic statement building**. When you type certain Visual Basic elements, Microsoft Access automatically tries to assist you in writing code by

displaying a drop-down list of appropriate choices for the code element you've typed. For example, if you type an object variable, and follow it with a period to indicate that you intend to enter a method or property, Microsoft Access automatically displays a list of the methods and properties that apply to the object. . . . . To complete the statement you're typing, you can either click an item in the list or continue typing your code. If you continue typing code, the list displays the closest match to what you've typed. To enter the selected item in the list at any time, press TAB. To make the list disappear, press ESC."

"Building Applications" is combinable with Antis and Coad for the reasons stated in Section V.A.13. (*See also* Ex. 1015, ¶ 230.)

I. Claim 13 Is Obvious Over Antis in view of Coad and Corda (U.S. Patent No. 5,758,122)

Claim 13 depends from claim 12, which, as discussed above, is obvious over Antis in view of Coad. Corda discloses the additional limitations in claim 13, which accordingly is obvious over Antis in view of Coad and Corda.

13. The integrated development environment as recited in claim 12, wherein the visualizer cooperates with the parser layer to change the appearance of displayed flows as a function of the source code segments determined to have errors.

Corda discloses this claim element. (*See* Ex. 1015, ¶¶ 231-235.) Corda discloses an "immersive visual programming environment"

that "permits the flow of data objects and the interaction among the data objects to be visually displayed to the user." (Corda at Abstract.) Corda further discloses that if an error is detected in source code, the programming environment changes the appearance of the displayed flow:

"Once the programmer is satisfied that the program as constructed is complete, compilation provides visual and/or audio clues as to the nature and location of compiler errors." (Corda at col. 4, lines 30-32.)

"Semantic error detection is enhanced through observation of the flow of data and the motion of the structure itself. In this way, the virtual programming environment of the present invention supports verification and validation. Run-time error messages in the form of sight and sound alerts are provided, with the offending component highlighted in some way." (*Id.* at col. 4, lines 41-48.)

"A method for programming . . . , said method comprising the steps of . . . compiling the computer program while providing visual and audio clues to a user in response to a detected compiler error, said clues indicating a location and nature of the detected compiler error . . . ." (*Id.* at col. 13, lines 13-22.)

Corda is combinable with Antis and Coad for the reasons stated in Section V.A.11. (*See also* Ex. 1015, ¶ 236.)

J. Claim 14 Is Obvious Over Antis in view of Coad and Access 97 Visual Basic

Claim 14 depends from claim 12, which, as discussed above, is obvious over Antis in view of Coad. Access 97 Visual Basic discloses the additional limitations in claim 14, which accordingly is obvious over Antis in view of Coad and Access 97 Visual Basic.

14. The integrated development environment as recited in claim 12, wherein the editor cooperates with the parser layer to change the appearance of portions of the displayed source code as a function of the software segments determined to have errors.
Access 97 Visual Basic discloses this claim element. (*See* Ex. 1015, ¶¶ 237-241.) At page 140, Access 97 Visual Basic discloses that code determined to have an errors is highlighted or shown in red text: "Now that you've cleared away that message, you'll notice that the '<' symbol is highlighted – this is the point in the line that Microsoft Access doesn't like – and the line is still shown in red text."

Access 97 Visual Basic is combinable with Antis and Coad for the reasons stated in Section V.A.7. (*See* Ex. 1015, ¶ 242.)

K. Dependent Claims 2-16 Are Unpatentable Over Each Reference and Combination of References Asserted for Claim 1 in View of the Prior Art

To the extent that they are not explicitly enumerated above, Petitioner hereby asserts grounds of unpatentability for dependent claims 2-16 that are premised upon the respective prior art references

asserted for those dependent claims in view of the following ground of unpatentability for independent claim 1 asserted above: Claim 1 is obvious over Coad in view of Oracle Primer and Oracle8 Primer.

(*See* Ex. 1015, ¶¶ 243-244.)

VI. Mandatory Notices Pursuant to 37 C.F.R. § 42.8(a)(1)

Pursuant to 37 C.F.R. § 42.8(a)(1), the mandatory notices identified in 37 C.F.R. § 42.8(b) are provided below as part of this Petition.

A. C.F.R. § 42.8(b)(a): Real Party-In-Interest

SAS is the real party-in-interest for Petitioner.

B. C.F.R. § 42.8(b)(2): Related Matters

The '936 patent is currently the subject of a patent infringement lawsuit brought by the assignee of the '936 patent, Complementsoft LLC, against SAS, captioned *ComplementSoft, LLC v. SAS Institute Inc.*, Docket No. 1:12-cv-07372 (N.D. Ill. Sept 14, 2012). This judicial matter may affect, or be affected by, decisions made in this proceeding.

C. C.F.R. § 42.8(b)(3) and (4): Lead and Back-up Counsel and Service Information

SAS provides the following designation of counsel:

Lead Counsel	Back-up Counsel
David B. Cochran	John V. Biernacki
Reg. No. 39,142	Reg. No. 40,511
JONES DAY	JONES DAY
901 Lakeside Avenue	901 Lakeside Avenue

Cleveland, Ohio 44114 (216) 586-7029 dcochran@jonesday.com	Cleveland, Ohio 44114 (216) 586-7747 jvbiernacki@jonesday.com John A. Marlott Reg. No. 37,031 JONES DAY 77 West Wacker Suite 3500 Chicago, Illinois 60601-1692 (312) 269-4236 jamarlott@jonesday.com

Pursuant to 37 C.F.R. § 42.10(b), a Power of Attorney accompanies this Petition. Please address all correspondence to lead and back-up counsel at the address above. SAS also consents to electronic service by email at the email addresses listed above.

VII. Conclusion

For at least the reasons set forth above, Petitioner requests *inter partes* review of the '936 patent because it is more likely than not that at least one of the claims challenged in this Petition is unpatentable. It is therefore respectfully submitted that this Petition be granted and claims 1-16 of the '936 patent be judged invalid. Counsel for the Petitioner may be contacted at the below listed telephone number. As identified in the attached Certificate of Service and in accordance with §§

1.33(c), 42.205, and 42.300, a copy of the present Request, in its entirety, is being served on the patent owner at the correspondence address of record for the subject patent as reflected in the publicly-available records of the United States Patent and Trademark Office as designated in the Office's Patent Application Information Retrieval system.

The Director is hereby authorized to charge any deficiency in the fees filed, asserted to be filed or which should have been filed herewith (or with any paper hereafter filed in this proceeding by this firm) to Jones Day Deposit Account No. 50-1432, ref: 343355-610009.

Respectfully submitted,

Date: March 29, 2013

/John V. Biernacki/
John V. Biernacki
Registration No. 40,511
JONES DAY
North Point, 901 Lakeside Avenue
Cleveland, Ohio 44114-1190

Direct No. (216) 586-7747

CERTIFICATE OF SERVICE

The undersigned hereby certifies that a copy of the foregoing Petition for *Inter Partes* Review of U.S. Patent No. 7,110,936, along with all exhibits supporting and filed with the Petition, were served on March 29, 2013, via Express Mail delivery directed to the attorney of record for the patent at the following address:

Barry Horwitz
Greenberg Traurig, LLP
77 West Wacker Drive
Suite 3100
Chicago, IL 60601-1732

Date: March 29, 2013

/John V. Biernacki/
John V. Biernacki
Registration No. 40,511
JONES DAY
North Point, 901 Lakeside Avenue
Cleveland, Ohio 44114-1190

EXHIBIT LIST

EXHIBIT NO.	TITLE
1001	U.S. Patent No. 7,110,936 ("the '936 Patent")
1002	Excerpt from the file history of the '936 Patent: May 31, 2005 Office Action
1003	Excerpt from the file history of the '936 Patent: August 4, 2005 Response to Office Action
1004	Excerpt from the file history of the '936 Patent: August 1, 2006 Supplemental Notice of Allowability
1005	U.S. Patent No. 5,572,650 ("Antis")
1006	U.S. Patent No. 6,851,107 ("Coad")
1007	U.S. Patent No. 6,356,285 ("Burkwald")
1008	U.S. Patent No. 5,937,064 ("Eick")
1009	"Microsoft Access 97 Visual Basic Step by Step" ("Access 97 Visual Basic")
1010	U.S. Patent No. 5,758,122 ("Corda")
1011	"Building Applications with Microsoft Access 97" ("Building Applications")
1012	Oracle Programming—A Primer ("Oracle Primer")
1013	Oracle8 Programming: A Primer ("Oracle8 Primer")

1015	Declaration of Dr. Nick Roussopoulos Under 37 C.F.R. § 1.68 in Support of Petition for *Inter Partes* Review of U.S. Patent No. 7,110,936
1016	Complementsoft's LPR 2.5 Initial Response to SAS Institute's Invalidity Contentions, *ComplementSoft, LLC v. SAS Institute Inc.*, Docket No. 1:12-cv-07372 (N.D. Ill. Sept 14, 2012)
1017	Declaration of Morris M. Jackson Under 37 C.F.R. § 1.68
1018	Pearson Customer Service Website
1019	Outgoing Email Message from Morris M. Jackson
1020	Email Message Received by Morris M. Jackson
1021	"Advanced System Editor, Enhanced P-System Editor," Joel Pitt, Infoworld, September 26, 1983
1022	"Encyclopedia of Computer Science, Fourth Edition," Anthony Ralston et al. eds., Nature Publishing Group 2000
1023	"IBM Dictionary of Computing," George McDaniel ed., McGraw-Hill 1994
1024	"The New IEEE Standard Dictionary of Electrical and Electronics Terms, Fifth Edition," Christopher J. Booth ed., The Institute of Electrical and

	Electronics Engineers, Inc. 1993
1025	"DB2 Update," July 1999
1026	"dBase is Loaded," Jan L. Harrington, MacUser, February 1, 1988
1027	"FoxPro 2.5 for Macintosh," Jeffrey Sullivan, MacUser, June 1, 1994
1028	"McGraw-Hill Dictionary of Scientific and Technical Terms," Fifth Edition, McGraw-Hill, Inc., New York, 1994
1029	"Use Cases in Object-Oriented Software Development," Mehmet Aksit, et al., AMIDST, February 5, 1999
1030	"Conformance Testing of Object-Oriented Components Specified by State/Transition Classes," Leonard Gallagher, National Institute of Standards and Technology, Information Technology Laboratory, May 19, 1999
1031	"Flowcharts, State Transition Tables and State Transition Diagrams," Herbert J. Bernstein, 1999
1032	"An ASM Semantics for UML Activity Diagrams," Egon Borger et al., Springer-Verlag Berlin Heidelberg, June 2000
1033	"Modeling Systems With UML," A Popkin Software White Paper, 1998
1034	"UML Notation Guide, Version 1.1,"

	September 1, 1997
1035	"Mole – A Java Based Mobile Agent System," Markus Straßer et al., Proceedings of the 2nd ECOOP Workshop on Mobile Object Systems, 1996
1036	"Saving Portable Computer Battery Power through Remote Process Execution," Alexey Rudenko et al., Mobile Computing and Communications Review, Volume 2, Number 1, January 1998
1037	"Better Template Error Messages," Andrei Alexandrescu, March 1999, available at http://erdani.com/publications/better_template_error_messages.html
1038	"CAP: An Automated Self-Assessment Tool to Check Pascal Programs for Syntax, Logic, and Style Errors," SIGCSE '95 Proceedings of the Twenty-Sixth SIGCSE Technical Symposium on Computer Science Education, 1995
1039	"Automatic Correction of Syntax Errors in Programming Languages," Jean-Pierre Levy, Cornell University, Thesis, December 1971
1040	"IEEE 100, The Authoritative Dictionary of IEEE Standards Terms," Seventh Edition, IEEE Press, 2000

Supreme Court of the United States
Office of the Clerk
Washington, DC 20543-0001

Scott S. Harris
Clerk of the Court
(202) 479-3011

May 22, 2017

Mr. Gregory A. Castanias
Jones Day
51 Louisiana Avenue, N.W.
Washington, DC 20001

Re: SAS Institute Inc.
v. Michelle K. Lee, Director, United States Patent and Trademark Office, et al.
No. 16-969

Dear Mr. Castanias,

The Court today entered the following order in the above-entitled case:

The petition for a writ of certiorari is granted.

Sincerely,
s/Scott S. Harris
Scott S. Harris, Clerk

Supreme Court, U.S.
FILED
JUL 20 2017
OFFICE OF THE CLERK

No. 16-969

IN THE

Supreme Court of the United States

SAS INSTITUTE INC.,

Petitioner,

v.

JOSEPH MATAL, Interim Director, U.S. Patent and Trademark Office, and COMPLEMENTSOFT, LLC,

Respondents.

On Writ of Certiorari to the United States Court of Appeals for the Federal Circuit

JOINT APPENDIX VOLUME II OF II

GREGORY A. CASTANIAS
Counsel of Record
JONES DAY
51 Louisiana Ave., NW
Washington, DC 20001
(202) 879-3939
gcastanias@jonesday.com

Counsel for Petitioner

JEFFREY B. WALL
Counsel of Record
DEPARTMENT OF JUSTICE
950 Pennsylvania Ave., NW
Washington, DC 20530
(202) 514-2217
supremectbriefs@usdoj.gov

Counsel for Respondents

PETITION FOR CERTIORARI FILED JANUARY 31, 2017
CERTIORARI GRANTED MAY 22, 2017

Library of Congress
Law Library

TABLE OF CONTENTS

Page

U.S. Patent No. 7,110,936 80

(12) **United States Patent**
Hiew et al.

(10) **Patent No.: US 7,110,936 B2**
(45) **Date of Patent: Sep. 19, 2006**

(54) **SYSTEM AND METHOD FOR GENERATING AND MAINTAINING SOFTWARE CODE**

(75) Inventors: **Fen Hiew**, Mendota Heights, MN (US); **Edwin M. Schroeder**, Chicago, IL (US)

(73) Assignee: **Complementsoft LLC**, Chicago, IL (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 923 days.

(21) Appl. No.: **09/992,624**

(22) Filed: **Nov. 19, 2001**

(65) **Prior Publication Data**

US 2005/0229154 A1 Oct. 13, 2005

Related U.S. Application Data

(60) Provisional application No. 60/270,950, filed on Feb. 23, 2001, provisional application No. 60/293,854, filed on May 25, 2001.

(51) **Int. Cl.**
G06F 9/45 (2006.01)

(52) **U.S. Cl.** **703/22**; 703/26; 717/100; 717/108; 717/134; 717/135

(58) **Field of Classification Search** 703/22; 717/108, 100, 134, 135; 345/700
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,187,788 A 2/1993 Marmelstein
5,485,615 A 1/1996 Wennmyr

(Continued)

FOREIGN PATENT DOCUMENTS

EP 1001338 A2 5/2000

(Continued)

OTHER PUBLICATIONS

"Language Independent Generation of graphical Representations of Source Code", Hendrix et al, ACM 0-89791-737-5, ACM 1995.*

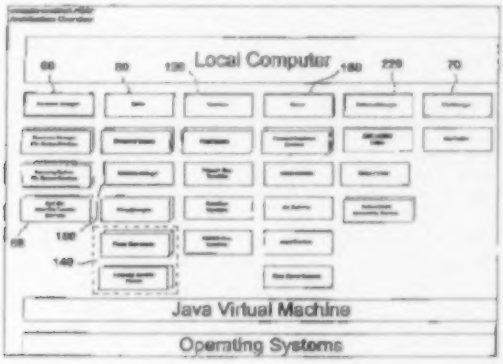
(Continued)

Primary Examiner—Fred Ferris
(74) *Attorney, Agent, or Firm*—Gary R. Jarosik

(57) **ABSTRACT**

A system and method for intelligently generating computer code. The system being comprised of a local computer, which is connected to a remote computer via a network system or the Internet and which is capable of exchanging files with the remote computer. The local computer is further comprised of a document manager for transferring files between the local computer and the remote computer and for providing enhanced file management functions. The document manager works in connection with the server module, the site manager and the connectivity layer to connect to remote computers, to transparently exchange files with the remote computer and to manage server profiles and connection information that is related to remote computers and transferred files. Once the file is transferred to the local computer, the editor can modify the code associated with the file; the editor is also capable of creating new files. The visualizer is capable of displaying a program flow diagram and a data flow diagram, which are comprised of program flow icons and data flow arrows to depict the code in terms of processing blocks and data blocks. To assist in developing new code or editing existing code, the template manager allows the user to browse through a directory of existing code sections or templates and to copy templates into the selected code for editing. For allowing the editor to process code that is written in different Data Management System programming languages and for creating the program flow icons, the parser layer detects the file type of a selected file and activates the rules and logic that apply to the corresponding Data Management System.

16 Claims, 42 Drawing Sheets



U.S. PATENT DOCUMENTS

5,652,899	A	7/1997	Mays et al.	
5,937,190	A *	8/1999	Gregory et al.	717/131
6,243,703	B1 *	6/2001	Couch et al.	707/10
6,311,323	B1	10/2001	Shulman et al.	
6,356,285	B1 *	3/2002	Burkwald et al.	715/853
6,604,110	B1	8/2003	Savage et al.	
6,851,107	B1 *	2/2005	Coad et al.	717/108
2001/0049682	A1 *	12/2001	Vincent et al.	707/100
2002/0059003	A1 *	5/2002	Ruth et al.	700/19
2002/0097253	A1 *	7/2002	Charisius et al.	345/700
2002/0112225	A1 *	8/2002	Charisius et al.	717/125
2002/0116702	A1 *	8/2002	Aptus et al.	717/170
2003/0041314	A1 *	2/2003	Heeren et al.	717/109
2003/0056192	A1 *	3/2003	Burgess	717/100
2003/0061600	A1 *	3/2003	Bates et al.	717/133

FOREIGN PATENT DOCUMENTS

WO	WO 97/40443	A1	10/1997
WO	WO 01/08007	A1	2/2001

OTHER PUBLICATIONS

"GENOA-A Customizable, Frount-End-Retargetable Source Code Analysis Framework", Devanbu, ACM Transactions on Software Engineering, vol. 8, No. 2, Apr. 1999.*

"Visualizing th Performance of Higher-Order Programs", Waddell et al., ACM 1-58133-055-04/98/0006, 1996 ACM.*

"Using a Fine-Grained Comparative Evaluation Technique to Understand and Design Software Visualization Tools", Mulholland, Seventh workshop on Empirical studies of programmers, ACM 1997.*

"An Analysis of Geometric Modeling in Database Systems", Kemper et al, ACM Computing Surveys, vol. 19, No. 1, Mar. 1987.*

Togethersoft Corporation, Together Documentation Set Version 4.2, Dec. 21, 2000, pp. 1-257.

Benedusi, P. et al., A Reverse Engineering Methodology to Reconstruct Hicarchical Data Flow Diagrams for Software Maintenance, Oct. 16-19, 1989, pp. 180-189

SAS Institute, SAS Companion for the Microsoft Windows Environment, Using the Enhanced Editor, Sep. 1999.

SAS Institute, Bibliographic and copyright information, SAS onlinedoc, version 8, Sep. 1999, 1page.

Anonymous, Togethersoft Ships Together Control Center 4.2, Latest Business Process Automation Software Speeds Development Time for E-Business Apps, Dec. 21, 2000, 2 pages.

SAS e-intelligence, Know Your Customers, SAS Institute Inc., 2001.

* cited by examiner

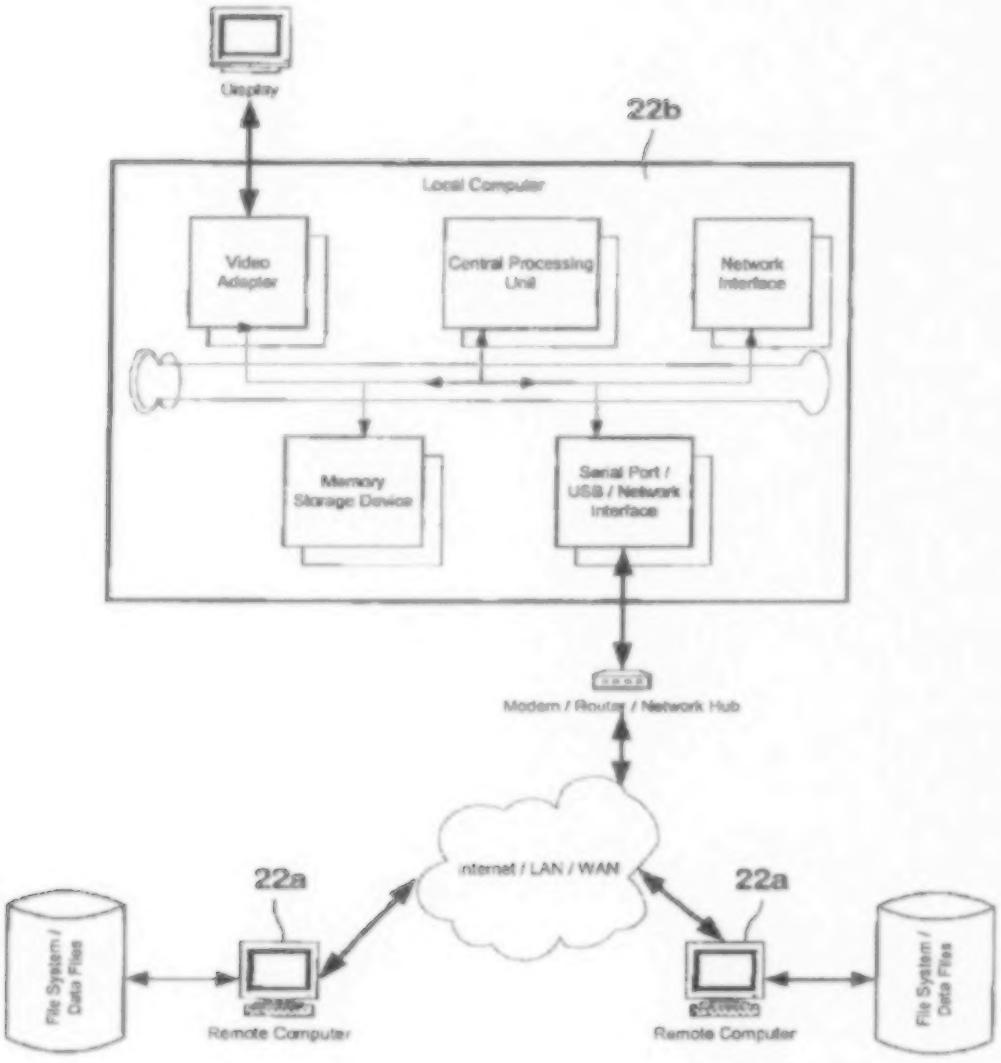

Display
22b
Local Computer
Video Adapter
Central Processing Unit
Network Interface
Memory Storage Device
Serial Port / USB / Network Interface
Modem / Router / Network Hub
Internet / LAN / WAN
22a
22a
File System / Data Files
Remote Computer
Remote Computer
File System / Data Files

FIG. 1

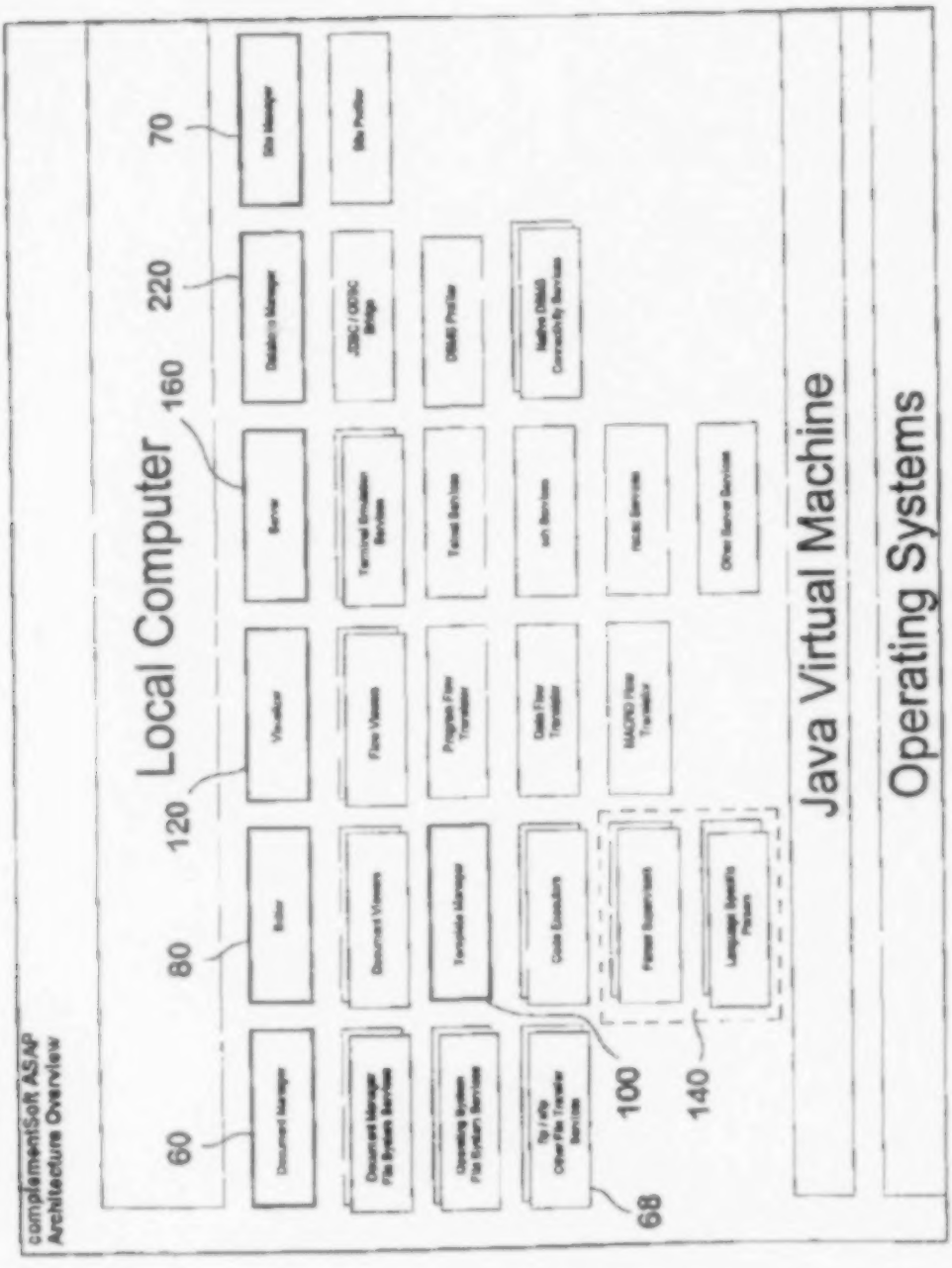
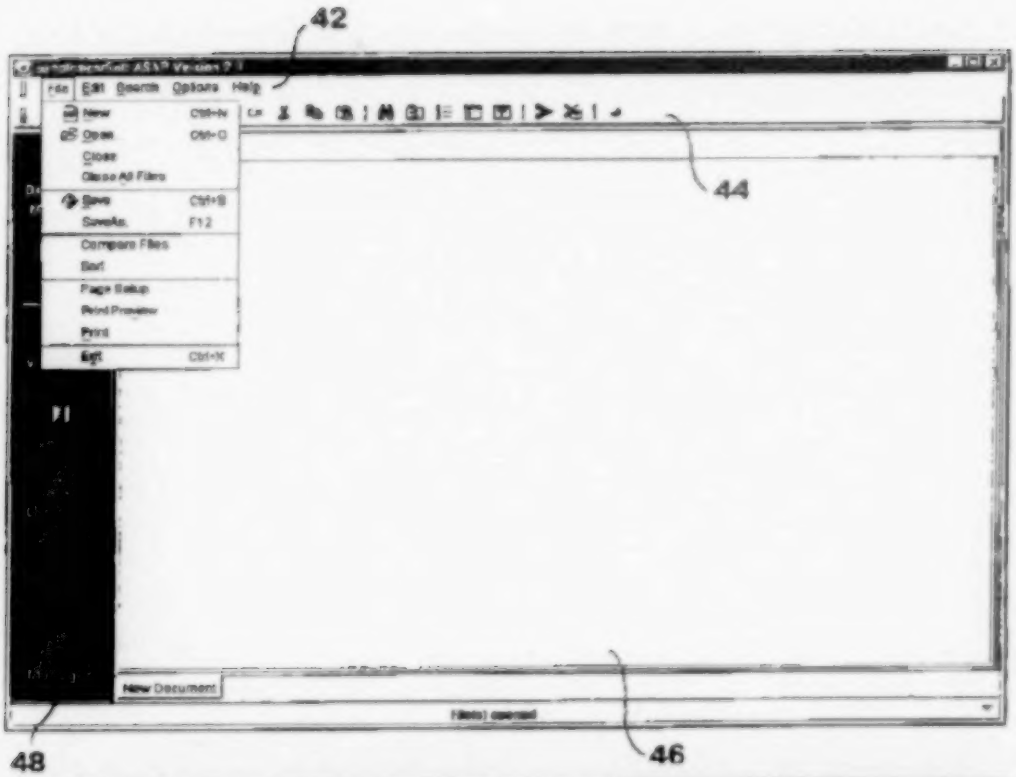

complementSoft ASAP
Architecture Overview
Local Computer
60
80
120
160
220
70
100
140
68
Java Virtual Machine
Operating Systems

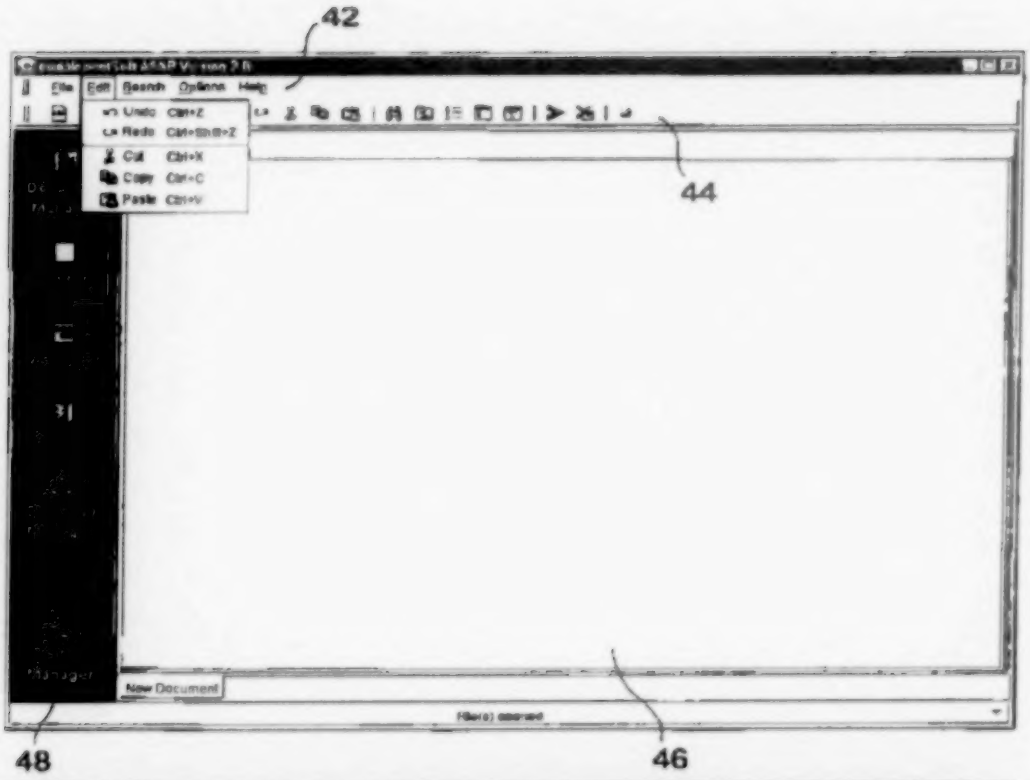
FIG. 2

FIG. 3



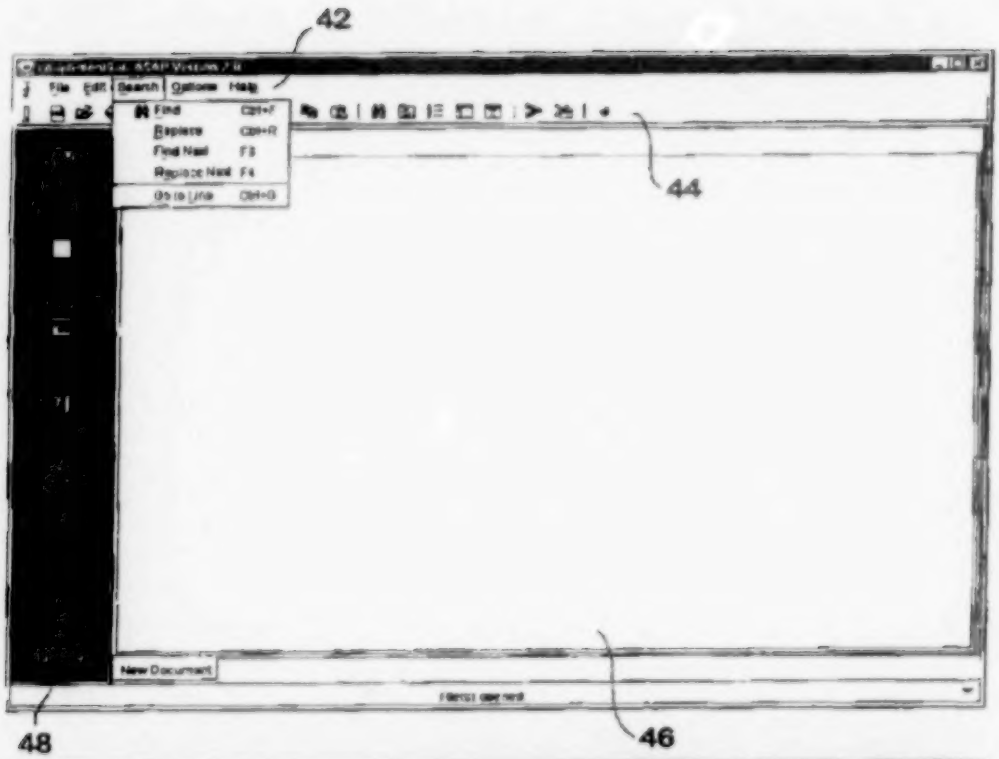
Menu Item	Functionality
New	Create a new document (a program)
Open	Open an existing document
Close	Close the currently opened document
Save	Save the currently open document
Save As	Save the currently open document using a different name or in a different location
Print Preview	Print Preview of the currently open (active) document
Print	Print the currently open (active) document
Exit	Exit the application

FIG. 4a



Menu Item	Functionality
Undo	
Redo	
Cut	
Copy	
Paste	

FIG. 4b



Menu Item	Functionality
Find	
Replace	
Find Next	
Replace Next	
Goto Line	Go to the n line in the currently active document

FIG. 4c

Menu Item	Functionality
Document Manager Configuration	Activate the Document Manager Configurator
Editor Configuration	Activate the Editor Configurator
Visual Configuration	Activate the Visual Configurator
Connection Configuration	Activate the Connection Configurator
Network File Configuration	Activate the Network File Configurator

FIG. 4d

Document Manager Configuration
General
Select the drive(s) to open in the local panel.
Local Drives:
A:\
C:\
D:\
E:\
OK
Cancel

FIG. 4da

Editor Configuration
General
Fonts
Color
Execute
84a
84b
84c
84d
84
Use external Editor
External Editor:
Disable line numbering
Print header information (Date File name Page number)
Print line numbers
Print wrapped text
Reset
OK
Cancel

FIG. 4db

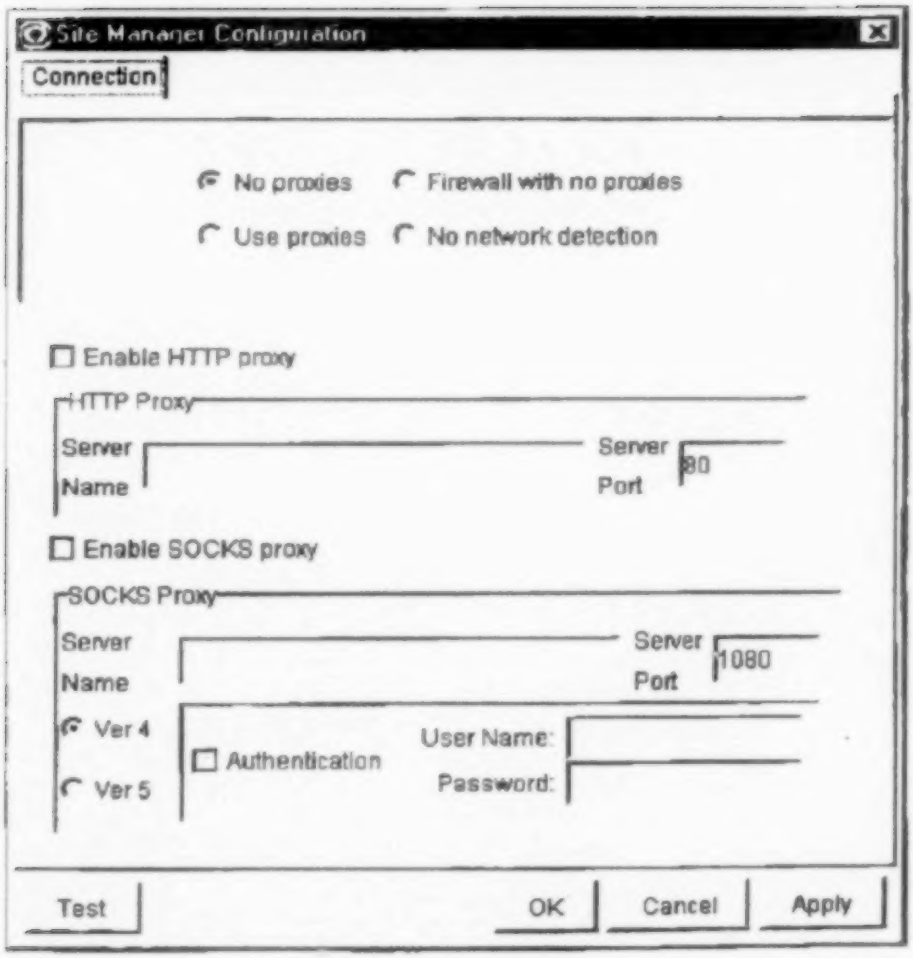

Site Manager Configuration
Connection
No proxies
Firewall with no proxies
Use proxies
No network detection
Enable HTTP proxy
HTTP Proxy
Server Name
Server Port
80
Enable SOCKS proxy
SOCKS Proxy
Server Name
Server Port
1080
Ver 4
Ver 5
Authentication
User Name:
Password:
Test
OK
Cancel
Apply

FIG. 4dc

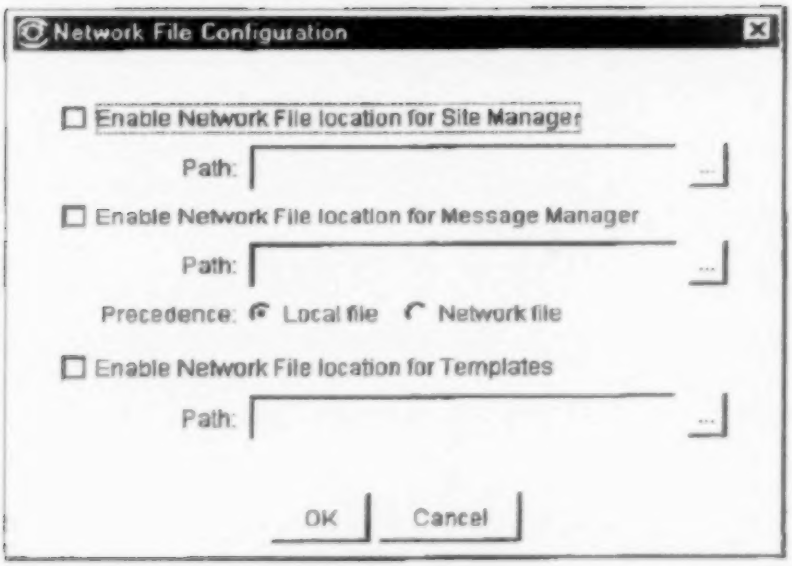
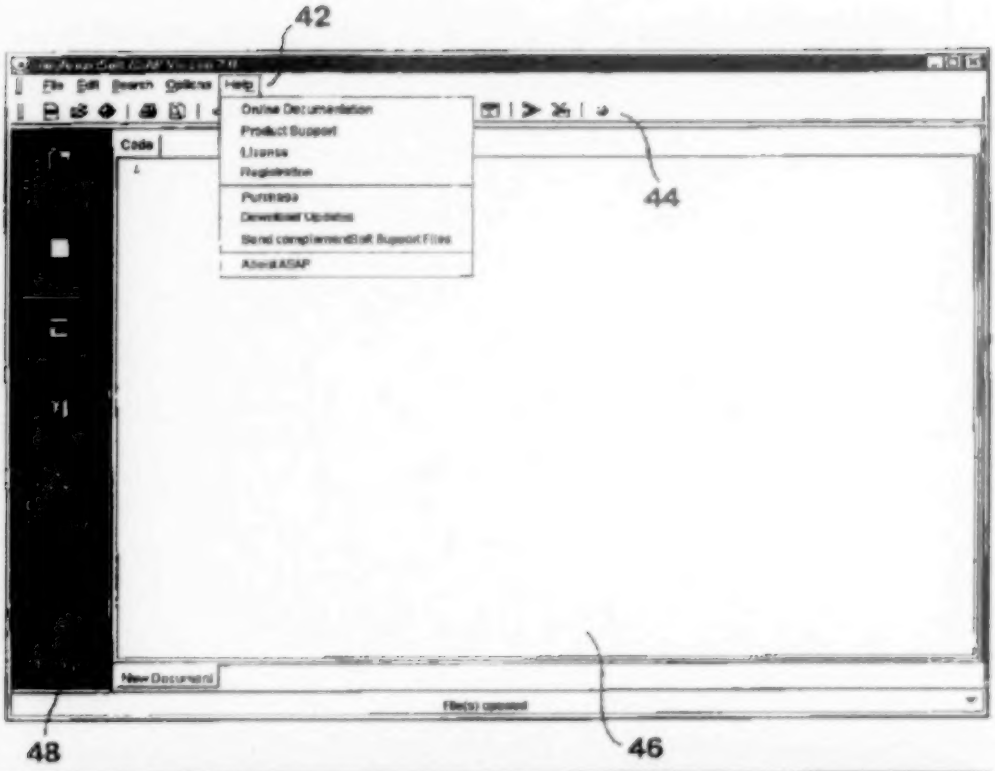

Network File Configuration
Enable Network File location for Site Manager
Path:
Enable Network File location for Message Manager
Path:
Precedence: Local file Network file
Enable Network File location for Templates
Path:
OK
Cancel

FIG. 4dd



Menu Item	Functionality
On-line Documentation	Show On-line Documentation
Product Support	Displays the Product Support Page with e-mail address, How to . . . and Frequently Asked Questions
License	Detailed license information with the ability to renew and/or upgrade the license
Registration	Perform on-line product registration
Purchase	Displays the complementSoft e-store
Download Updates	Retrieve Product Updates
Send complementsoft Support files	E-mail support files to complementSoft
About ASAP	Version and copyright information

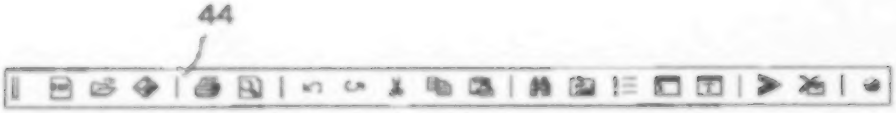
FIG. 4e

File Edit Search Options Help
Code
42
44
Redo
Cut
Copy
Paste
Find
Find Next
Replace
Replace Next
Close
Line numbering
Tree split
Templates
Edit Properties
Execute program
Toggle Break Points
Message Manager
New Document
48
46

FIG. 4f

Menu Item	Functionality
Undo	
Redo	
Cut	
Copy	
Paste	
Find	
Find Next	
Replace	
Replace Next	
Close	Close the currently active document
Line numbering	Turn line numbering on of off
Tree Split	Activate the Tree View split screen mode
Template Manager	Activate the Template Manager
Edit Properties	Edit the properties (timestamp, originating location, etc.) of the document
Execute Program	Execute the currently active program (i.e., the one that the user is currently editing) The user can further specify the execution mode (i.e., how to execute, with which option turn on/off, etc) and execution location (i.e., where to execute, which server, which execution engine, which location) If the user has previously activated Break-Point execution (see below), then the execution will be carried out based out the break-points, i.e., start at where the user indicate that it should start, end at where the user indicate that it should end and skipping statements as indicated by the user.
Toggle Break Points	Activate / deactivate Break-Point execution. Setup Break-Points if they have not been previously setup. Reset/Remove Break-Points if they have been previously setup
Message Manager	Activate the Message Manager to add, remove, or update the messages in the message repository

FIG. 4f (continued)



Menu Item	Functionality
New	
Open	
Save	
Print	
Print Preview	
Undo	
Redo	
Cut	
Copy	
Paste	
Find	Find a word
Refresh	Refresh the Editor display
Toggle Line Numbers	Show or hide line numbers
Toggle Tree View	Enable or disable Tree View
Template Manager	Activate the Template manager
Execute Program	Execute the currently active program
Close	Close the currently active program
Toggle Break Point Execution	Enable of disable Break Point Execution

FIG. 5

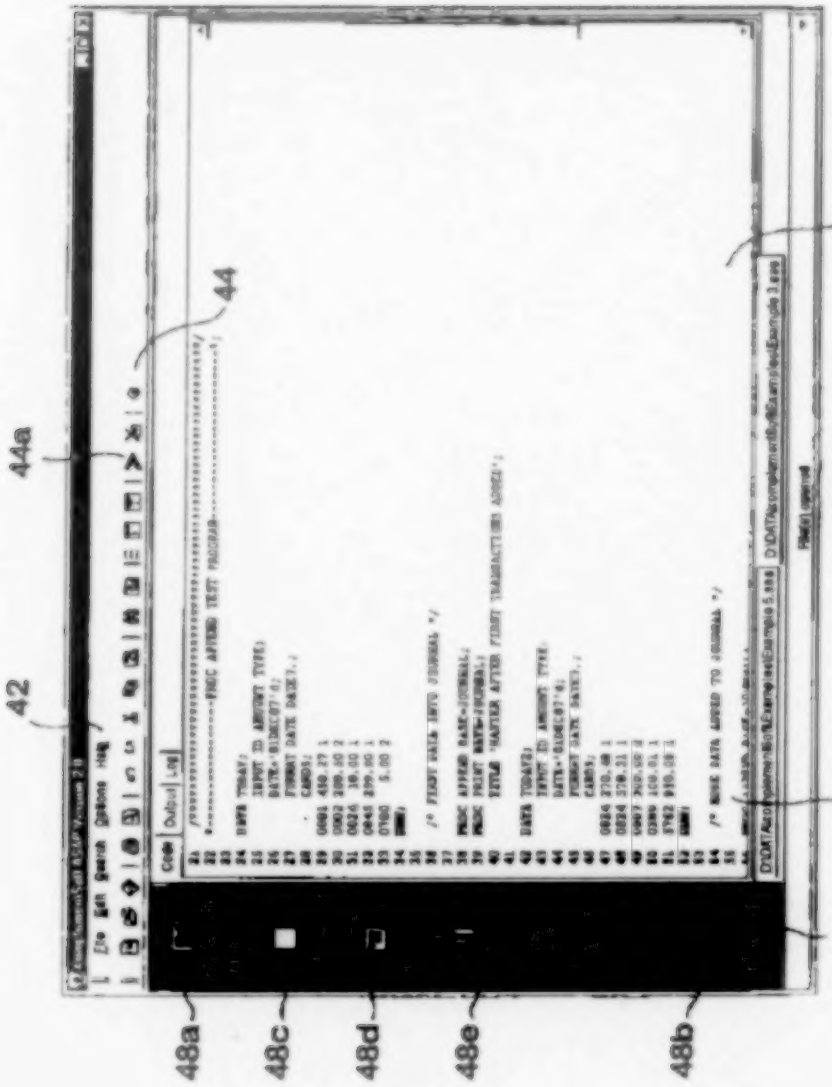


FIG. 6

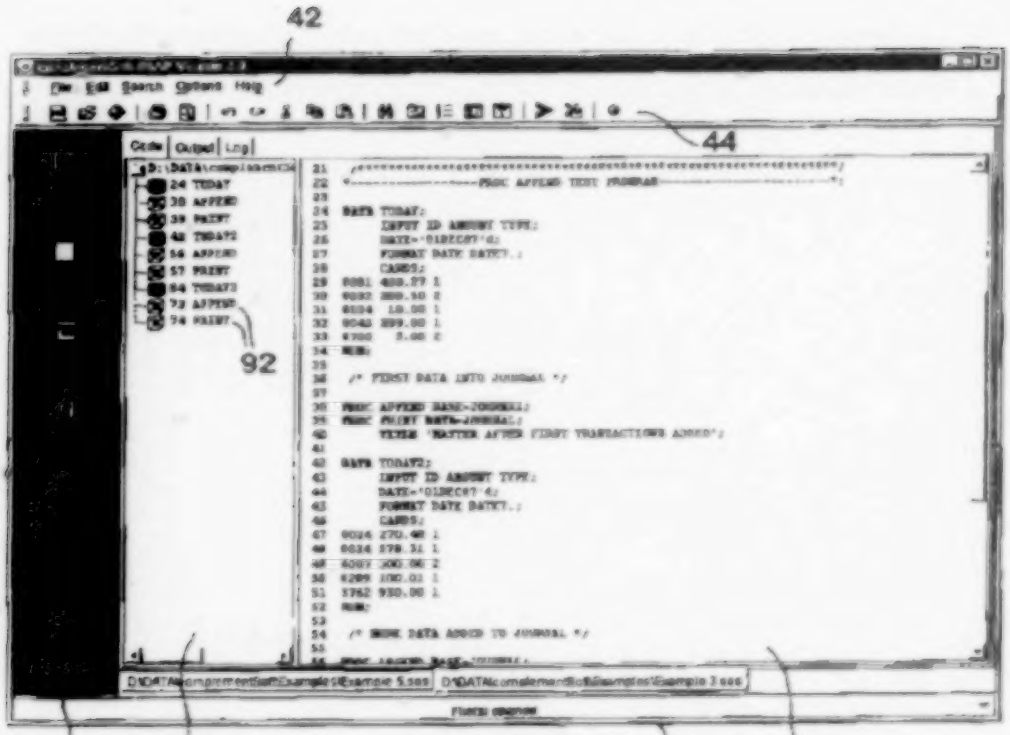


FIG. 7

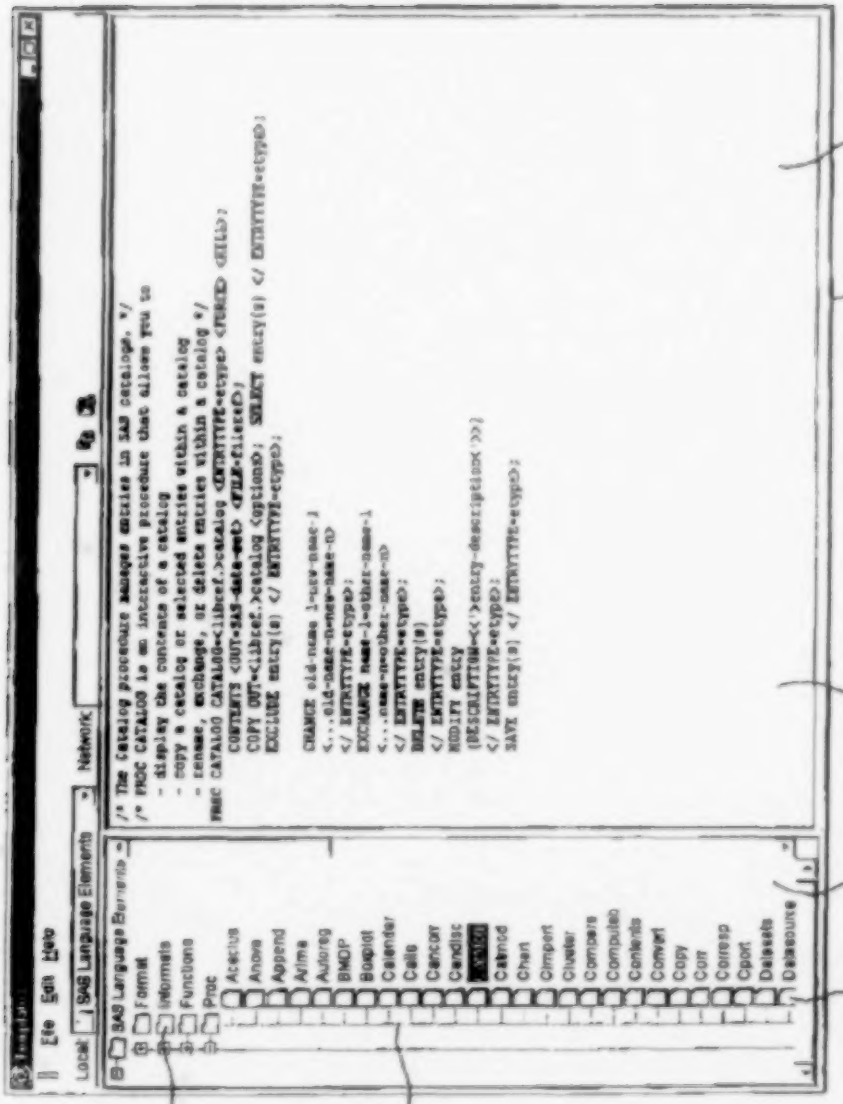

File Edit Help
Local
SAS Language Elements
Network
SAS Language Elements
Format
Informats
Functions
Proc
Datasets
Datasource
/* The Catalog procedure manages entries in SAS catalogs. */
/* PROC CATALOG is an interactive procedure that allows you to
- display the contents of a catalog
- copy a catalog or selected entries within a catalog
- rename, exchange, or delete entries within a catalog */
PROC CATALOG CATALOG=<libref.>catalog <ENTRYTYPE=etype> <FORCE> <KILL>;
CONTENTS <OUT=SAS-data-set> <FILE=fileref>;
COPY OUT=<libref.>catalog <options>; SELECT entry(s) </ ENTRYTYPE=etype>;
EXCLUDE entry(s) </ ENTRYTYPE=etype>;
CHANGE old-name-1=new-name-1
<...old-name-n=new-name-n>
</ ENTRYTYPE=etype>;
EXCHANGE name-1=other-name-1
<...name-n=other-name-n>
</ ENTRYTYPE=etype>;
MODIFY entry
</ ENTRYTYPE=etype>;
MODIFY entry
(DESCRIPTION=<'>entry-description<'>)
</ ENTRYTYPE=etype>;
SAVE entry(s) </ ENTRYTYPE=etype>;
105
104
106
101a
101b
100
101

FIG. 8

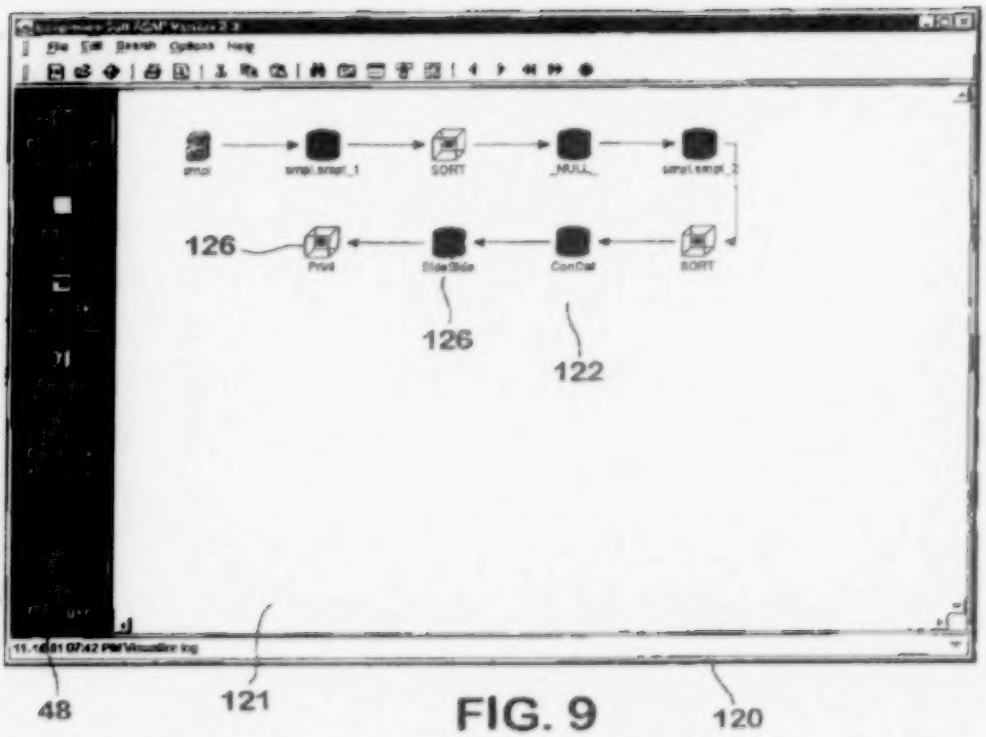


FIG. 9

File Edit Search Options Help
Server A Server B
Connected to 192.168.1.4 telnet
48
162
160

FIG. 10

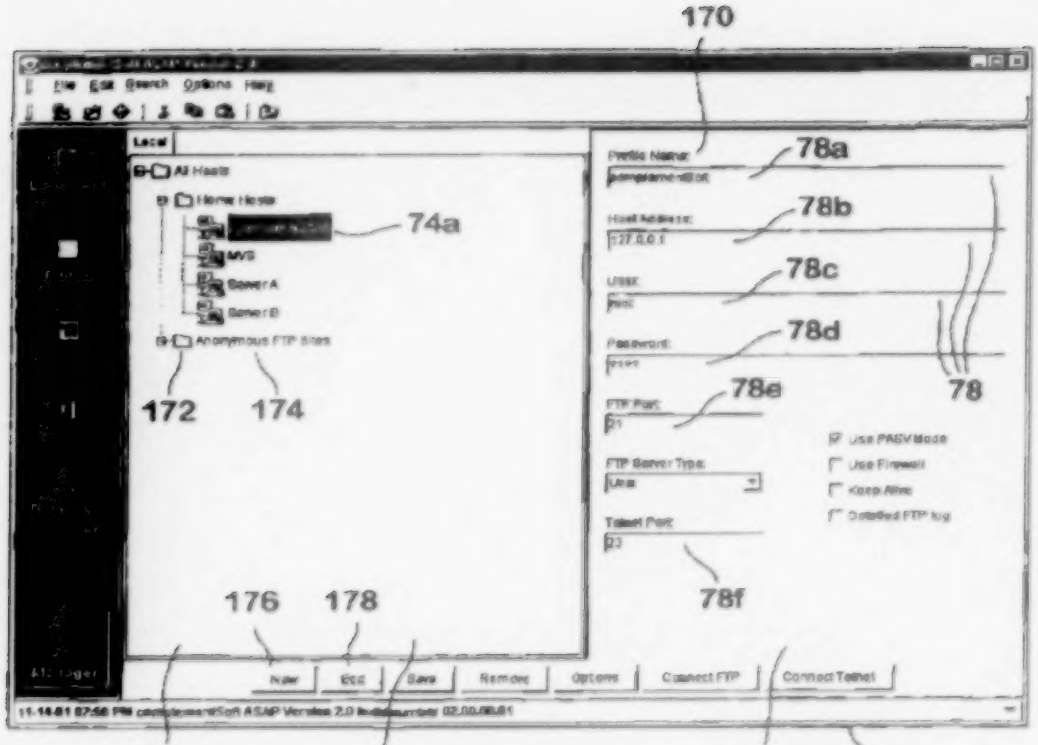


FIG. 11

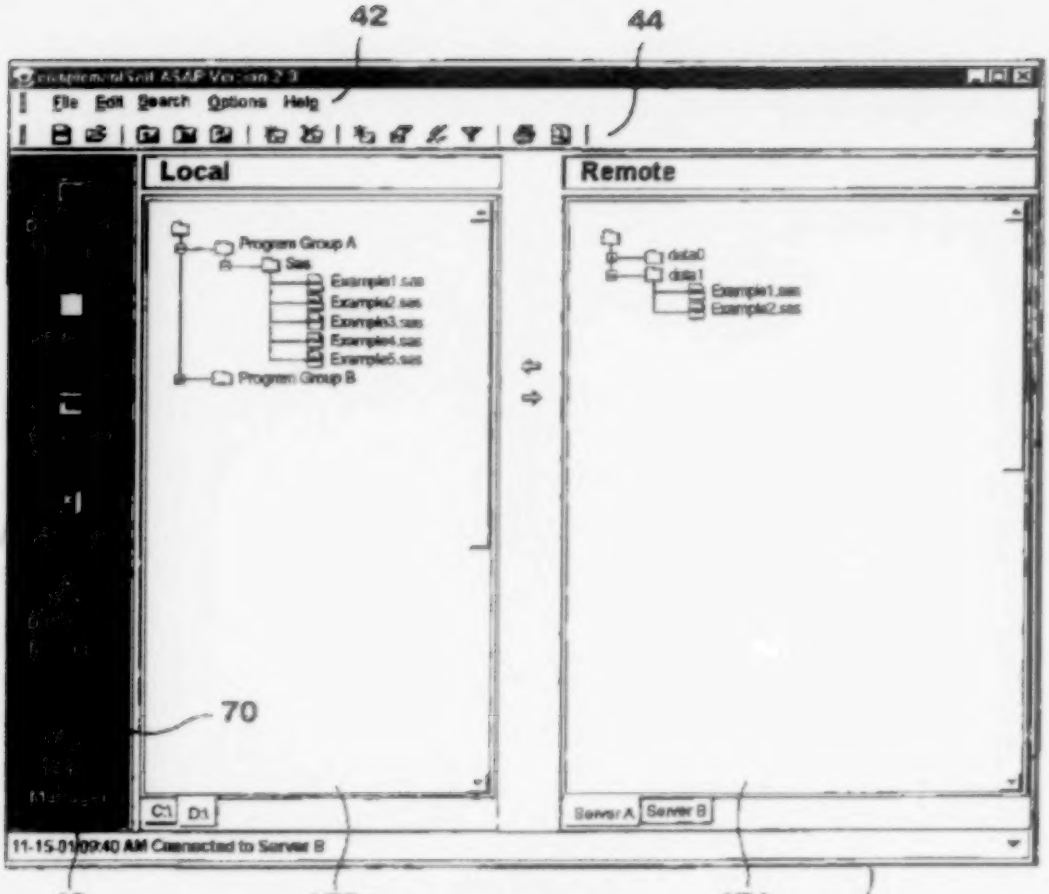

42
44
File Edit Search Options Help
Local
Remote
Program Group A
Sas
Program Group B
70
C:\ D:\
Server A Server B
48
172
174
60

FIG. 12

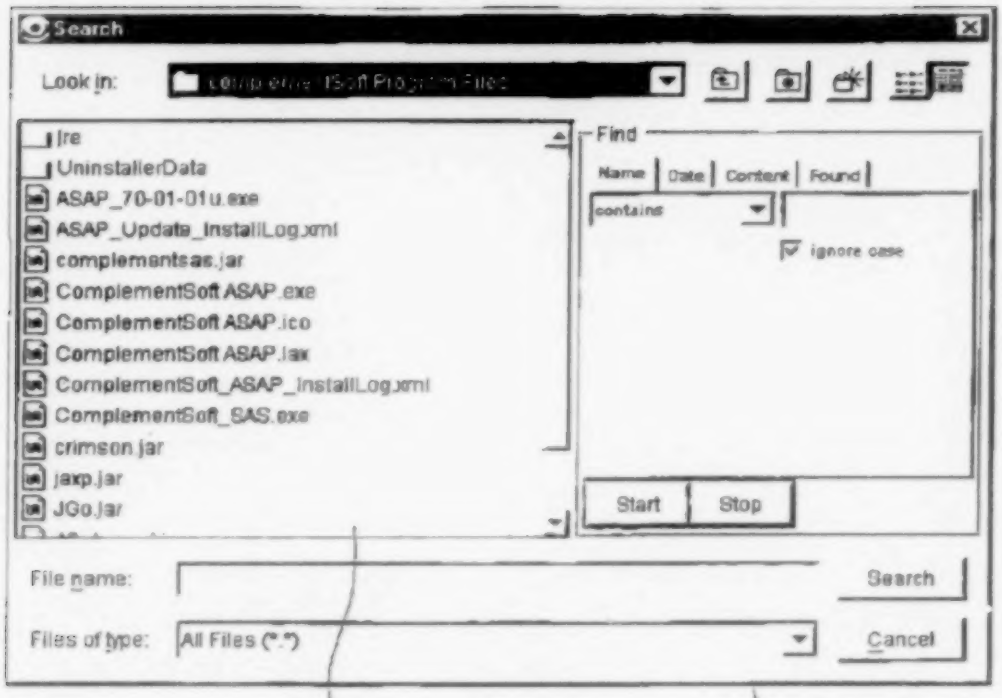

Search
Look in:
jre
UninstallerData
ASAP_70-01-01u.exe
ASAP_Update_InstallLog.xml
complementsas.jar
ComplementSoft ASAP.exe
ComplementSoft ASAP.ico
ComplementSoft ASAP.lax
ComplementSoft_ASAP_InstallLog.xml
ComplementSoft_SAS.exe
crimson.jar
jaxp.jar
JGo.jar
Find
Name
Date
Content
Found
contains
ignore case
Start
Stop
File name:
Search
Files of type:
All Files (*.*)
Cancel
69a
69

FIG. 13

FIG. 14a

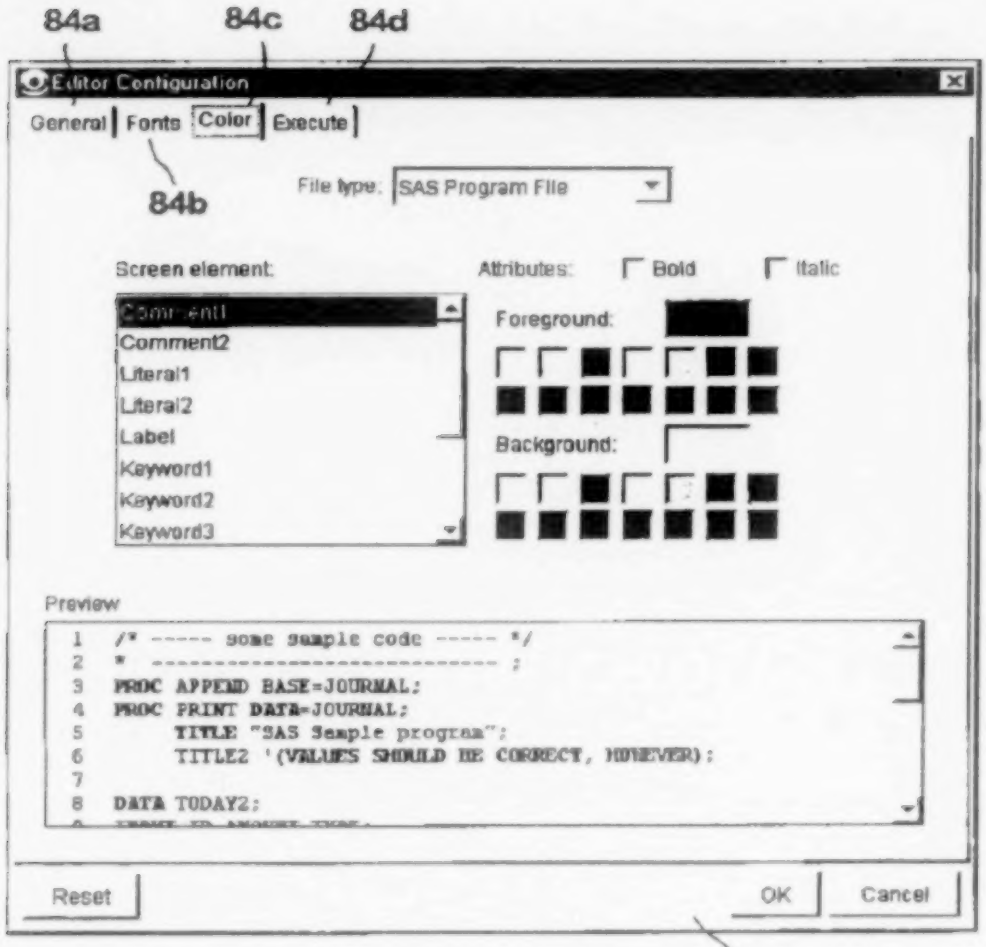

84a
84c
84d
Editor Configuration
General
Fonts
Color
Execute
84b
File type:
SAS Program File
Screen element:
Attributes:
Bold
Italic
Comment1
Comment2
Literal1
Literal2
Label
Keyword1
Keyword2
Keyword3
Foreground:
Background:
Preview
1 /* ----- some sample code ----- */
2 * ---------- ;
3 PROC APPEND BASE=JOURNAL;
4 PROC PRINT DATA=JOURNAL;
5 TITLE "SAS Sample program";
6 TITLE2 '(VALUES SHOULD BE CORRECT, HOWEVER);
7
8 DATA TODAY2;
Reset
OK
Cancel
84

FIG. 14b

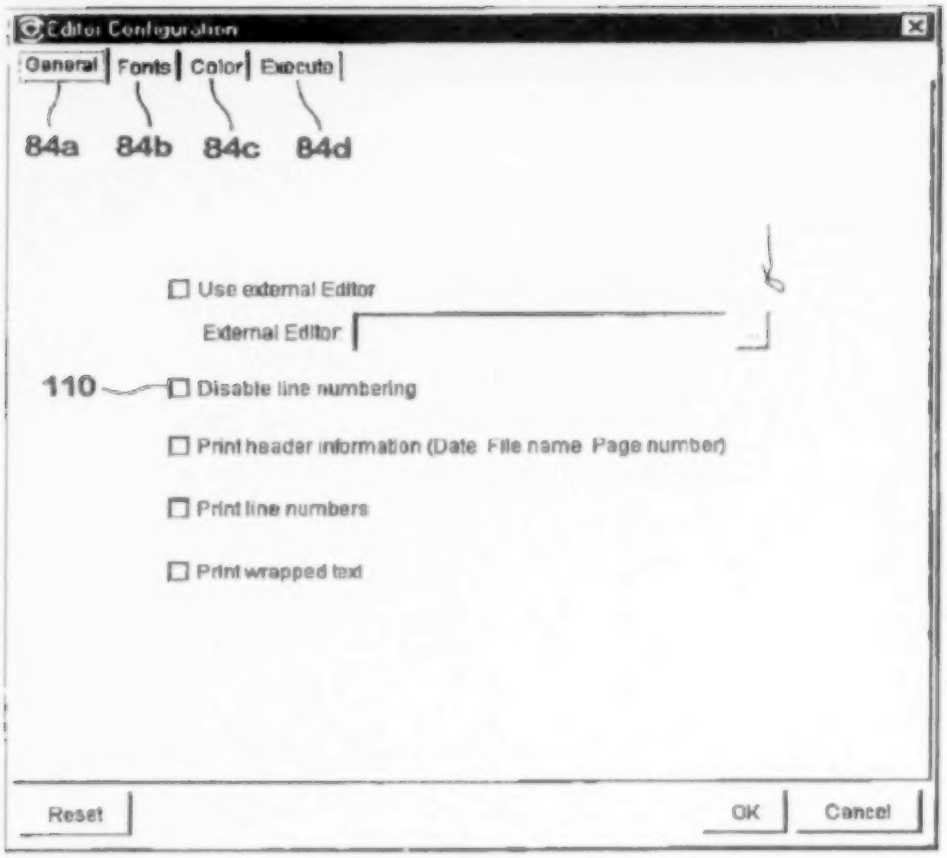

Editor Configuration
General
Fonts
Color
Execute
84a
84b
84c
84d
Use external Editor
External Editor
110
Disable line numbering
Print header information (Date File name Page number)
Print line numbers
Print wrapped text
Reset
OK
Cancel

FIG. 14c

84a
84d
Editor Configuration
General
Fonts
Color
Execute
Execution Mode
84b
84c
N Observations
Observations 0
Expand Macros (MPRINT)
Execution Location
Execution Path
Local
Program Files\SAS Institute\SAS\V8\SAS.exe
Remote
Don't show option dialog before execution.
Reset
OK
Cancel

FIG. 14d

84a
84b
84c
84d
General
Fonts
Color
Executs
Font Name:
Courier New
Size:
12
Style:
Plain
Preview
1 /* ----- some sample code ----- */
2 * ---------- ;
3 PROC APPEND BASE=JOURNAL;
4 PROC PRINT DATA=JOURNAL;
5 TITLE "SAS Sample program";
6 TITLE2 '(VALUES SHOULD BE CORRECT, HOWEVER);
7
8 DATA TODAY2;
9 INPUT ID AMOUNT TYPE;
10 DATE='01DEC87'd;
11 FORMAT DATE DATE7.;
12 CARDS;
13 LOOP:
14 0024 270.48 1
15 0024 578.31 1
16 0007 300.00 2
17 0289 100.01 1
18 5762 950.00 1
19
20 IF VPT_MER > 0 THEN
21 VPC_MER = VHA_MER / VPT_MER * 100;
Reset
OK
Cancel

FIG. 14e

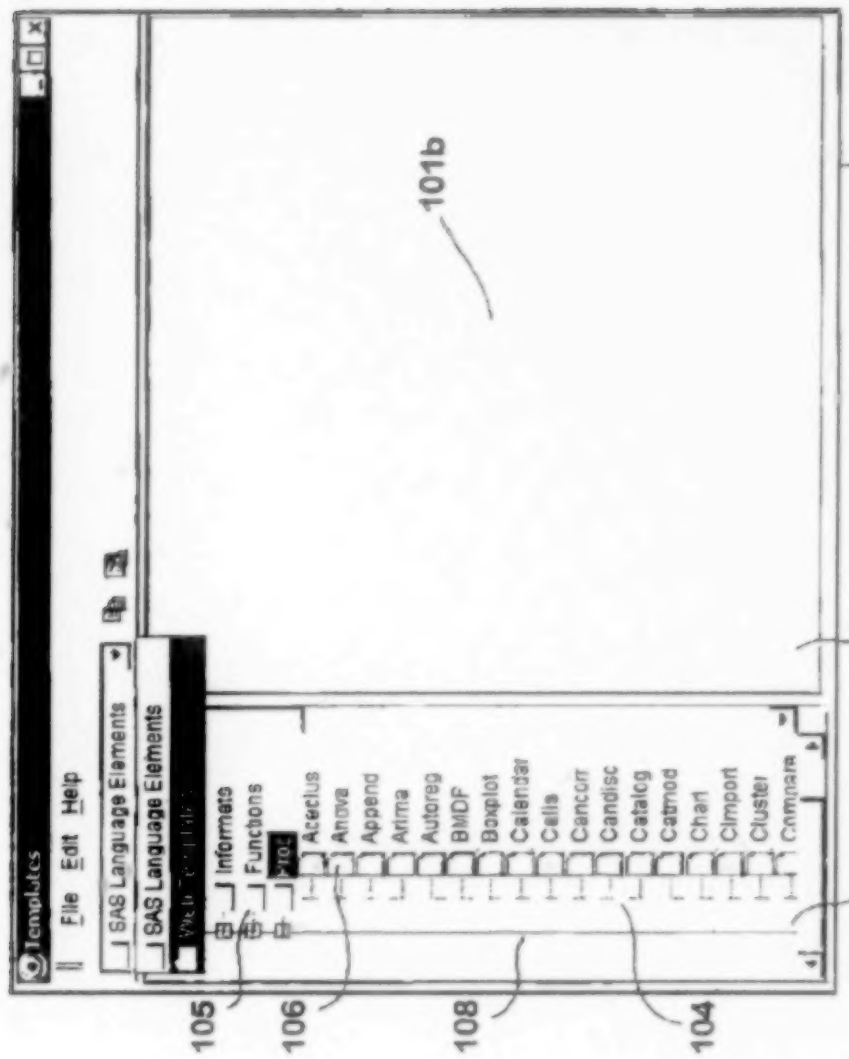

Templates
File Edit Help
SAS Language Elements
SAS Language Elements
Informats
Functions
Acecius
Anova
Append
Arima
Autoreg
BMDF
Boxplot
Calendar
Calis
Cancorr
Candisc
Catalog
Catmod
Chart
Cimport
Cluster
Commana
101b
100
101
101a
105
106
108
104

FIG. 15

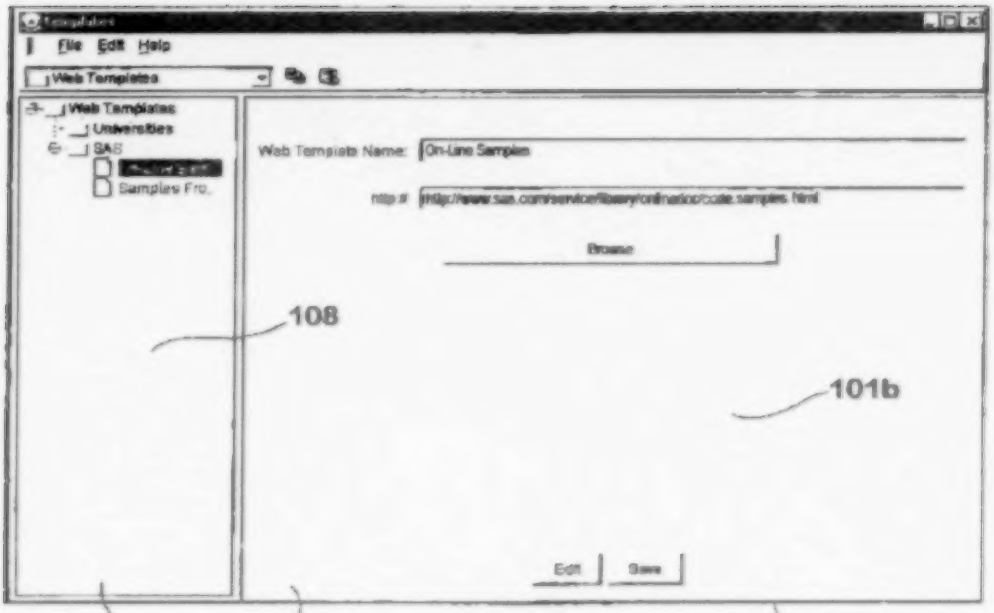

File Edit Help
Web Templates
Web Templates
Universities
SAS
Samples Fro..
Web Template Name:
On-Line Samples
http #
Browse
108
101b
Edit
Save
101a
101
100

FIG. 16

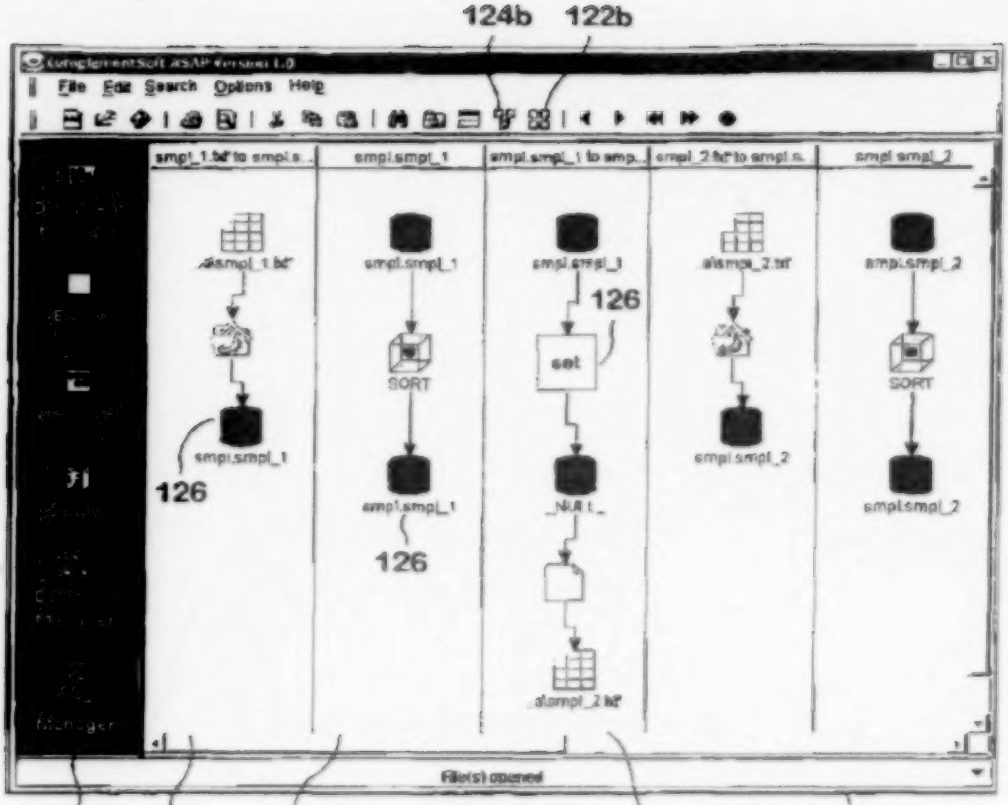

124b
122b
File Edit Search Options Help
smpl.smpl_1
SORT
126
set
NULL
File(s) opened
48
121
124
124a
120

FIG. 17

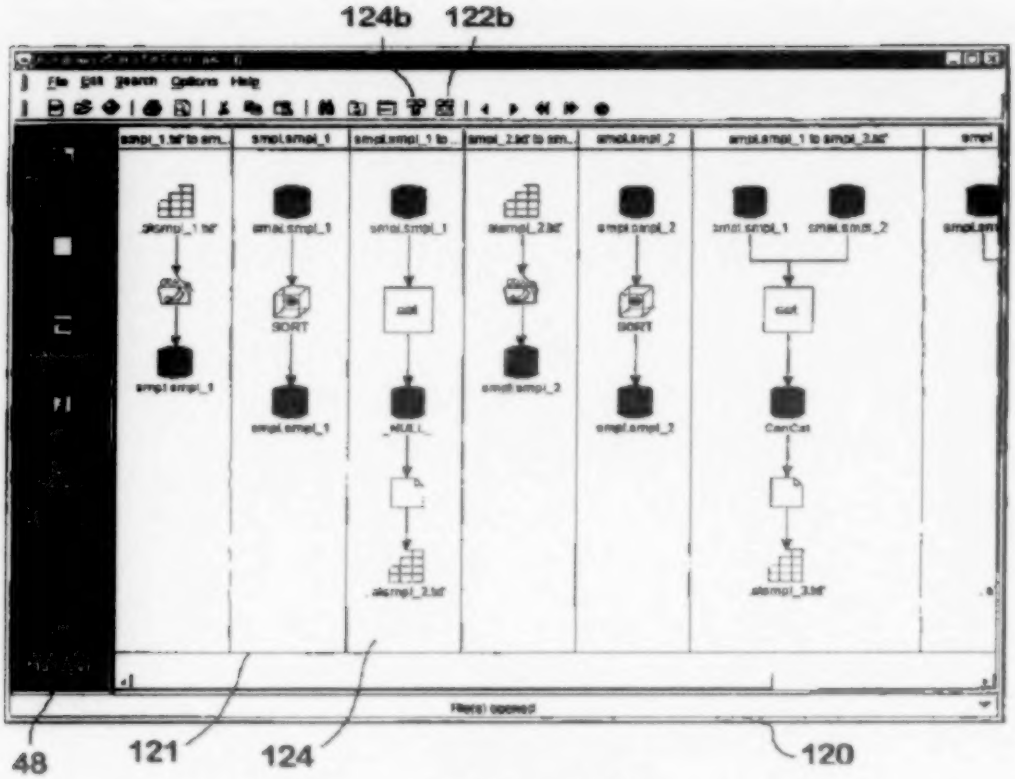


FIG. 18a

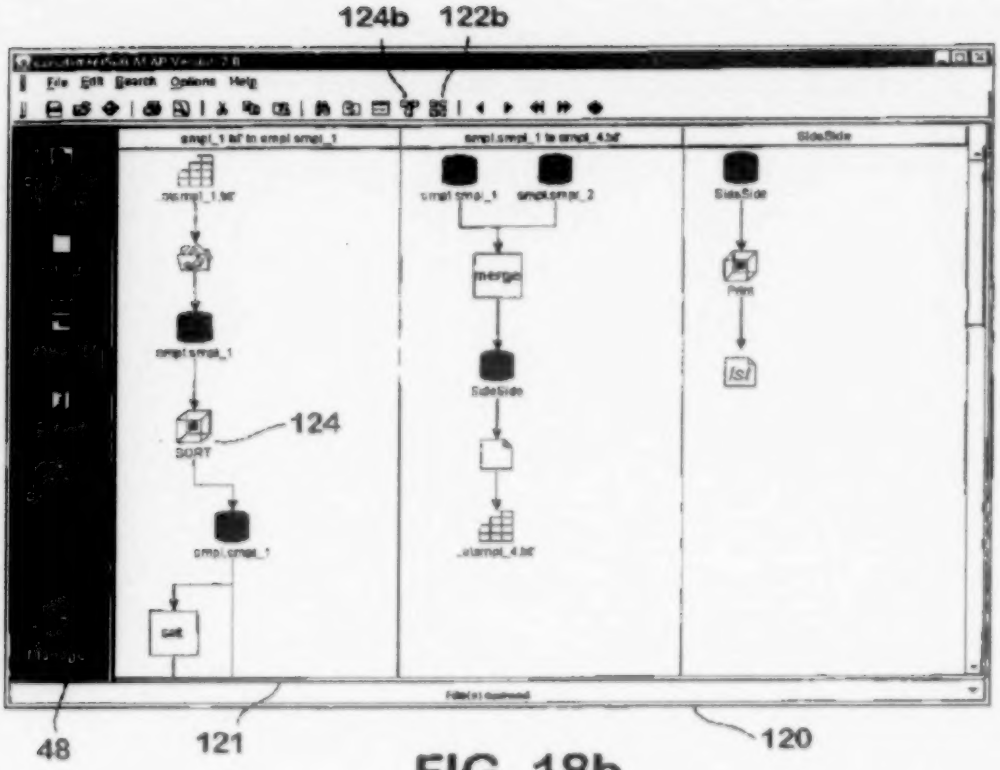


FIG. 18b

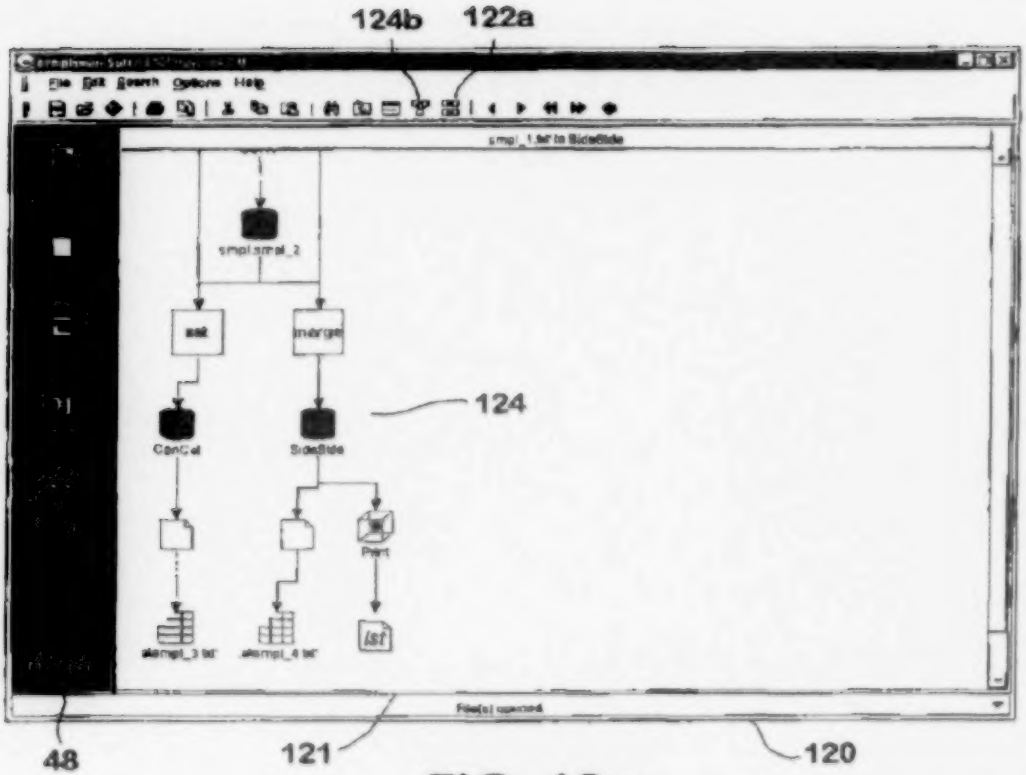


FIG. 18c

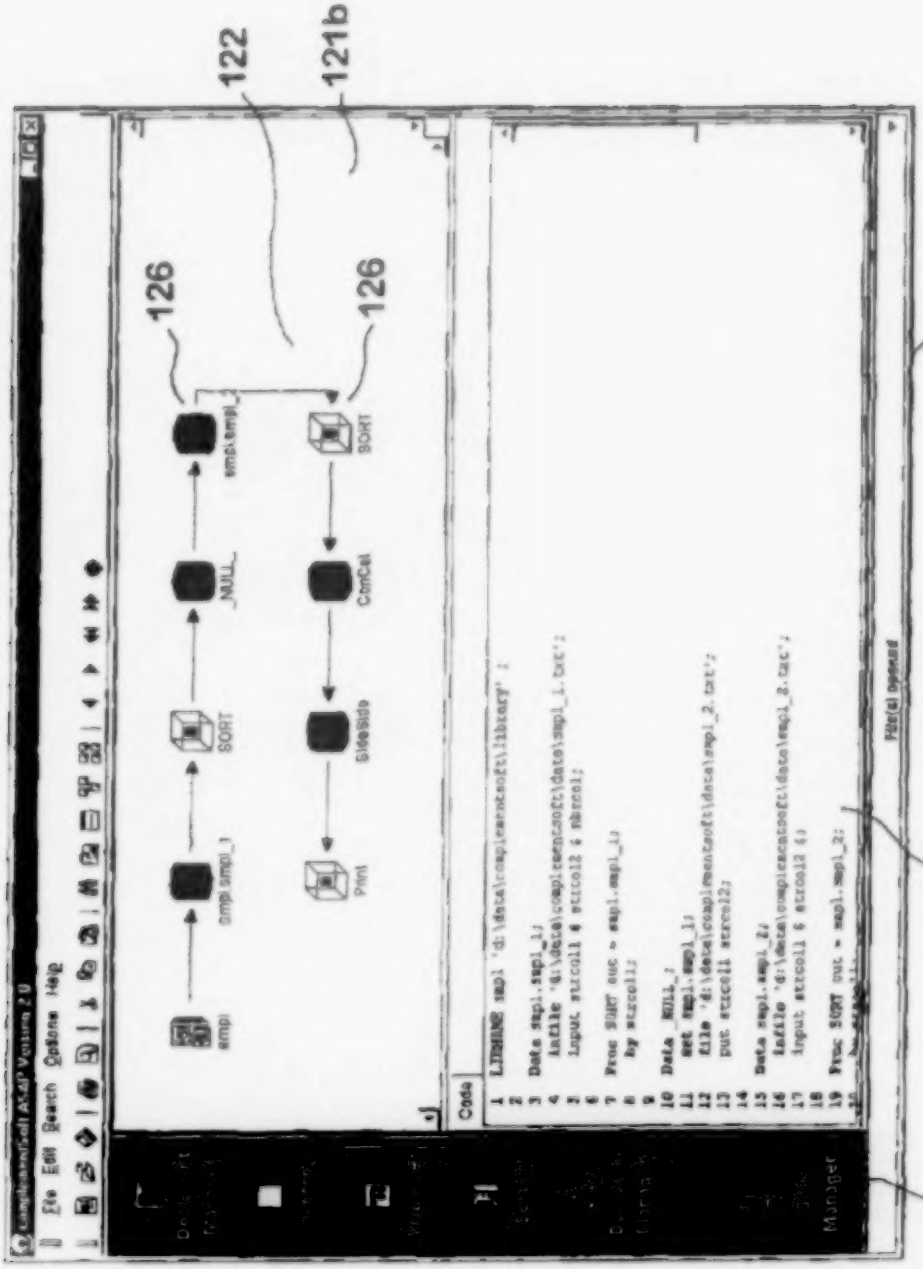

122
121b
126
126
120
121c
48
File Edit Search Options Help
empl
empl.empl_1
SORT
NULL
empl.empl_2
Print
BiteSide
ConCat
SORT
Code
Manager

FIG. 19

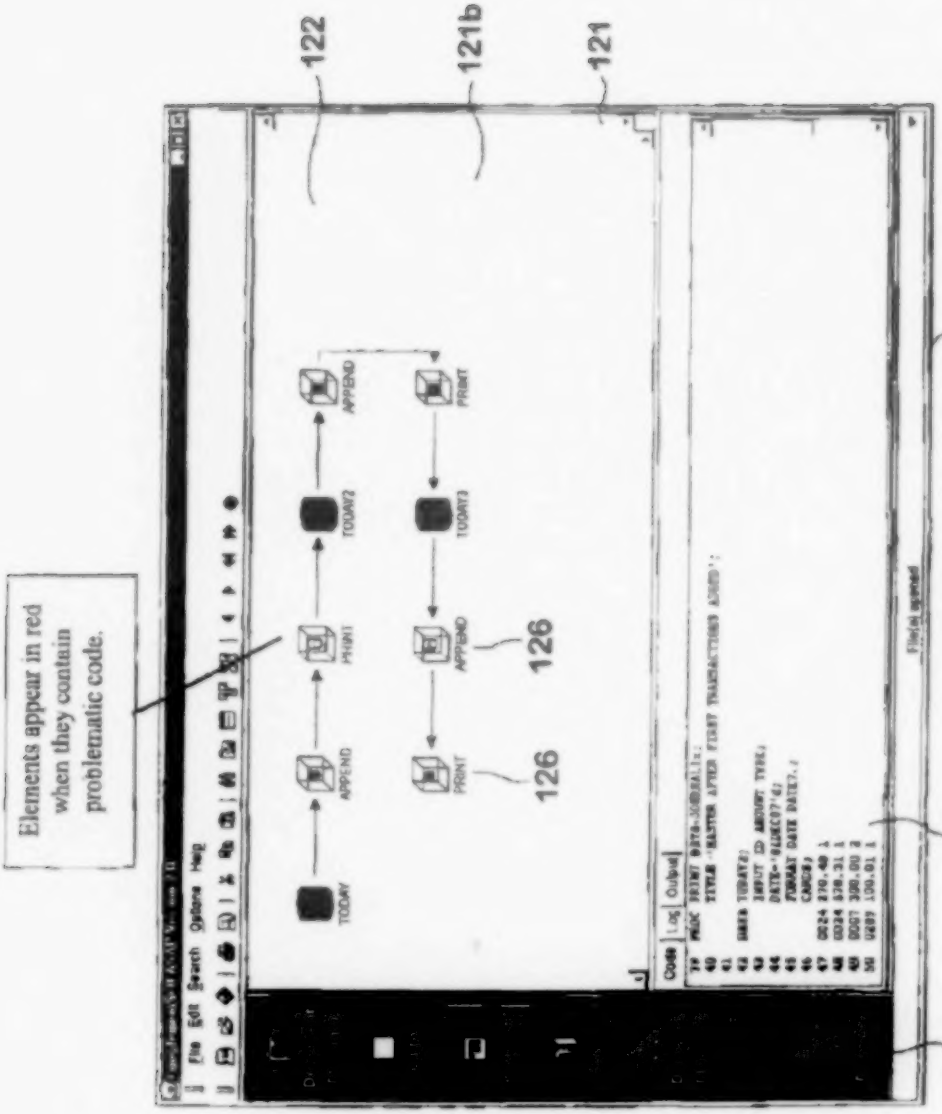

Elements appear in red when they contain problematic code.
122
121b
121
120
126
126
121
48
TODAY
APPEND
PRINT
TODAY2
APPEND
PRINT
TODAY3
Code
Log
Output

FIG. 20

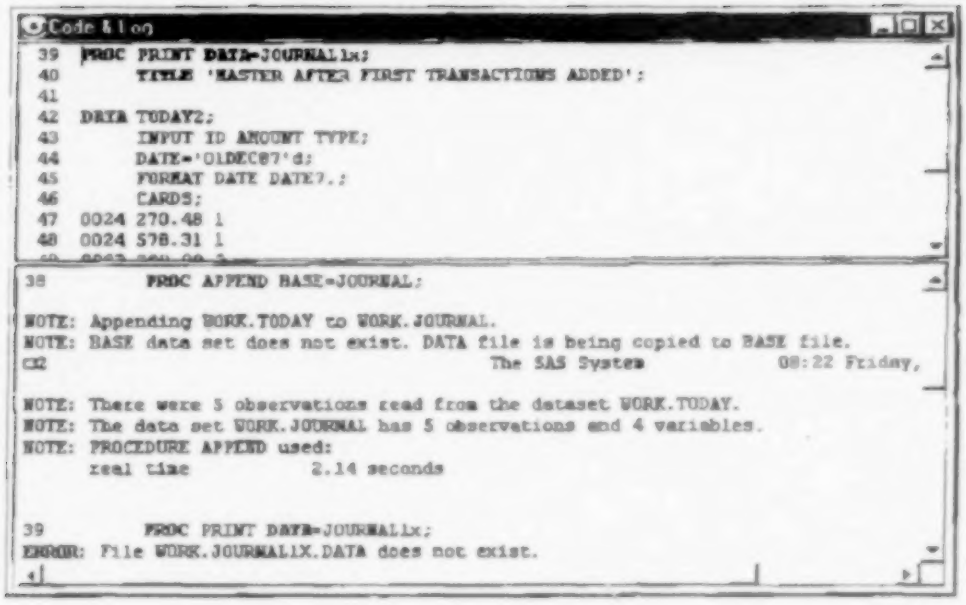

Code & Log
39 PROC PRINT DATA=JOURNAL1x;
40 TITLE 'MASTER AFTER FIRST TRANSACTIONS ADDED';
41
42 DATA TODAY2;
43 INPUT ID AMOUNT TYPE;
44 DATE='01DEC87'd;
45 FORMAT DATE DATE7.;
46 CARDS;
47 0024 270.48 1
48 0024 578.31 1
38 PROC APPEND BASE=JOURNAL;
NOTE: Appending WORK.TODAY to WORK.JOURNAL.
NOTE: BASE data set does not exist. DATA file is being copied to BASE file.
The SAS System 08:22 Friday,
NOTE: There were 5 observations read from the dataset WORK.TODAY.
NOTE: The data set WORK.JOURNAL has 5 observations and 4 variables.
NOTE: PROCEDURE APPEND used:
real time 2.14 seconds
39 PROC PRINT DATA=JOURNAL1x;
ERROR: File WORK.JOURNAL1X.DATA does not exist.

FIG. 20a

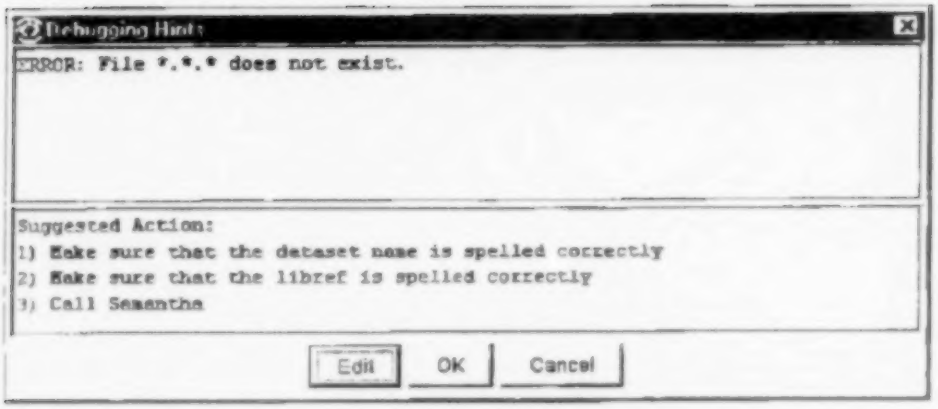

Debugging Hints
ERROR: File *.*.* does not exist.
Suggested Action:
1) Make sure that the dataset name is spelled correctly
2) Make sure that the libref is spelled correctly
3) Call Samantha
Edit
OK
Cancel

FIG. 20b

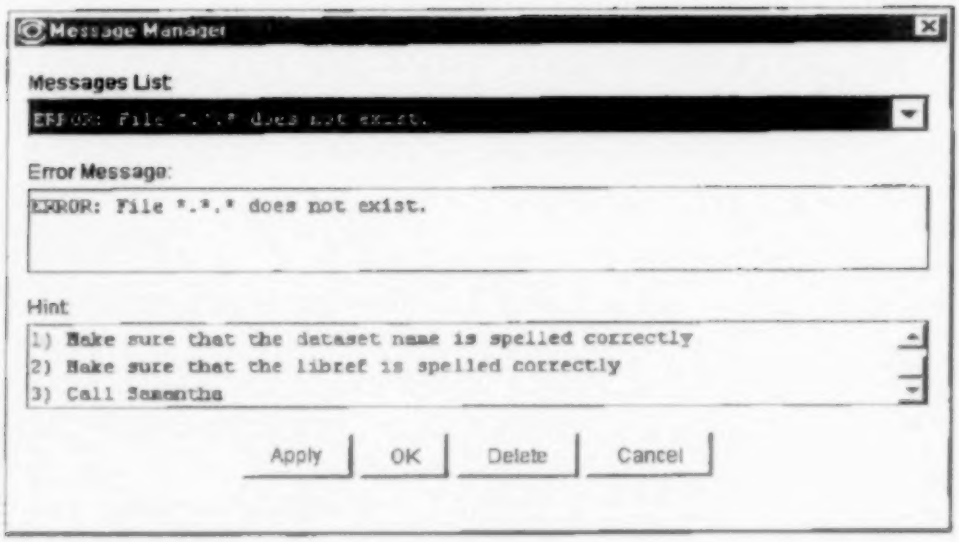

Message Manager
Messages List
ERROR: File *.*.* does not exist.
Error Message:
ERROR: File *.*.* does not exist.
Hint:
1) Make sure that the dataset name is spelled correctly
2) Make sure that the libref is spelled correctly
3) Call Samantha
Apply
OK
Delete
Cancel

FIG. 20c

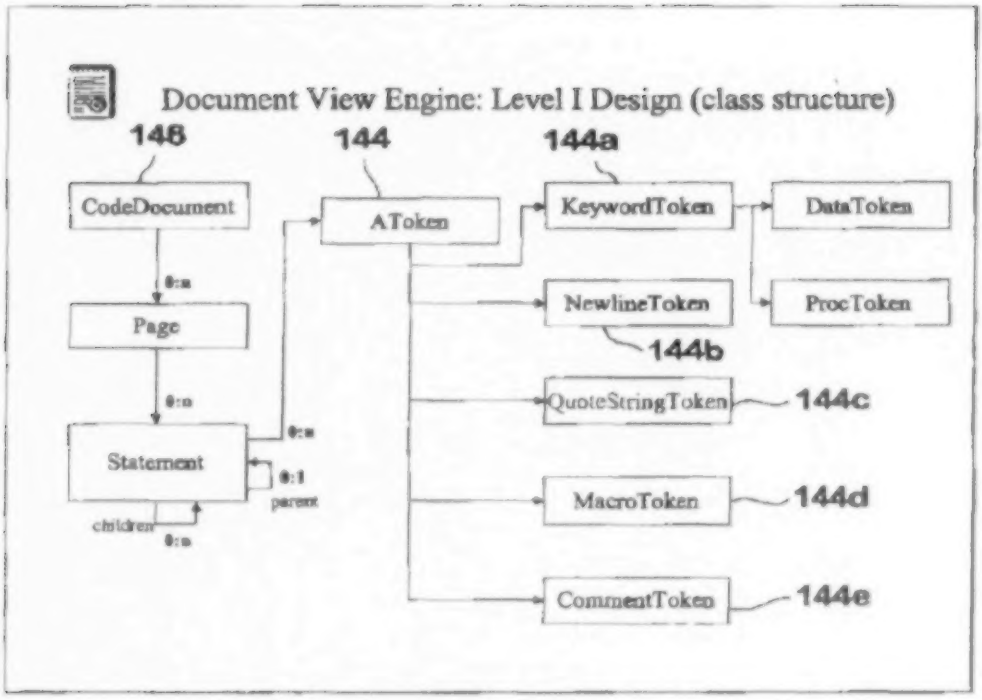

Document View Engine: Level I Design (class structure)
148
144
144a
CodeDocument
AToken
KeywordToken
DataToken
0:n
Page
NewlineToken
ProcToken
144b
0:n
QuoteStringToken
144c
0:n
Statement
0:1
parent
MacroToken
144d
children
0:n
CommentToken
144e

FIG. 21

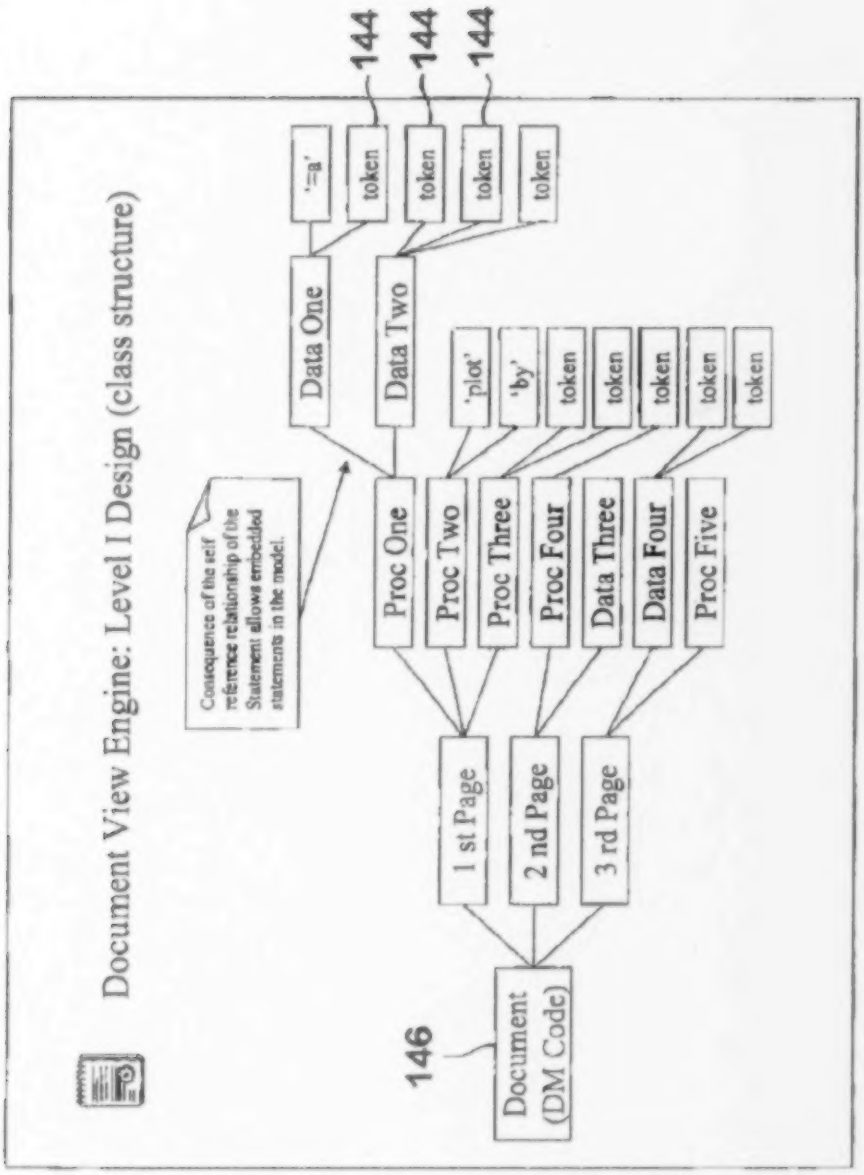

Document View Engine: Level I Design (class structure)
144
144
144
'=a'
token
token
token
token
Data One
Data Two
'plot'
'by'
token
token
token
token
token
token
Consequence of the self reference relationship of the Statement allows embedded statements in the model.
Proc One
Proc Two
Proc Three
Proc Four
Data Three
Data Four
Proc Five
1 st Page
2 nd Page
3 rd Page
146
Document (DM Code)

FIG. 22

FIG. 23

SYSTEM AND METHOD FOR GENERATING AND MAINTAINING SOFTWARE CODE

RELATED APPLICATION

This subject application claims the benefit of U.S. Provisional Application Ser. Nos. 60/270,950, entitled "GUI SAS CODE Development and Maintenance Environment Software," filed on Feb. 23, 2001 and 60/293,854, entitled "Integrated Development Environment and GUI for Data Management Systems," filed on May 25, 2001.

BACKGROUND OF THE INVENTION

This invention relates generally to software maintenance and development tools and, more particularly, to an extensible, language independent software development tool having a graphical user interface, i.e., a GUI Integrated Development Environment.

The evolution of data manipulation and data management systems, such as SAS®, SPSS® and SQL®, and relational database management systems, such as IBM® DB2 UDB® and the Oracle® RDBMS, has resulted in several high-level software languages that are inconsistent and, in some cases, unstructured. Based on these inconsistencies and the unstructured nature of some of these languages, database management languages may be difficult to use, edit and debug. Moreover, because of the lack of a standardized syntax among these programming languages, it has been difficult for users of these languages to share code. This is particularly the case with SAS®.

Since the advent of personal computers and the GUI Interfaces such as Windows® Interfaces, it has become increasingly necessary for businesses to develop or purchase customized software in order to support specific business strategies or processes. This, in turn, has led to the implementation of a variety of software development tools and Integrated Development Environments ("IDEs"). Generally, these software development tools assist users and programmers in editing, debugging and developing software for specific programming languages. Software providers of data manipulation languages or systems have, however, failed to provide a single comprehensive software development tool capable of assisting users in the editing, visualizing, debugging and development of software. Furthermore, the creation of graphical development tools for particular programming languages, such as, SAS®, has been inhibited by the intricacies of the programming language itself.

Over the years, point solutions have been introduced to address specific issues. For example, there are a few GUI based ftp software packages to make the job of transferring and managing code easier. There are also a few GUI based editor software packages to make editing code easier. Still further, there are software packages that one can use to manually diagram program flow and data flow. What is lacking, however, is a software package that integrates these best of breed point solutions and integrates them in such a way that they work seamlessly with each other, e.g., to have an editor that is integrated with a ftp package so that the editor can edit files that are located in a remote server or to have the editor integrated with the diagramming package so the user does not have to manually generate and update program flow or data flow diagrams.

Thus, a need exists for an Integrated Development Environment for generating and maintaining software code, in particular, for data manipulation centric languages. More specifically, a need exists for a system and method for

exchanging, editing, debugging, visualizing and developing SAS®, SPSS®, SQL®, DB2 UDB®, Oracle® RDBMS and other relational database management system software.

SUMMARY OF THE INVENTION

In accordance with these needs, the present invention is embodied in an Integrated Development Environment for generating and maintaining source code (software programmed in a software language), in particular, programmed in data manipulation languages. Generally, the system in which the Integrated Development Environment resides includes a local computer capable of exchanging files with a remote computer via a network system, i.e., a Local Area Network, a Wide Area Network, or the Internet. The local computer preferably hosts the Integrated Developers Environment which is further comprised of a document manager for transferring files and otherwise providing enhanced file management functions, such as, version synchronization across multiple platforms. The document manager works in connection with a server module, a site manager and a connectivity layer which is part of the Integrated Development Environment to connect to remote computers, to transparently exchange files with the remote computer and to manage server profiles and connection information that is related to remote computers and transferred files.

Once a file is transferred to the local computer, an editor, which is included as part of and integrated with the Integrated Development Environment, can modify the code associated with the file. In addition, the editor is also capable of creating new files and provides many advanced editing features such as visual execution break points, standardized formatting of files, and line numbering to name a few. A visualizer, i.e., a software tools that reads the code and generates diagrams and graphical representation of the program flow, data flow or the logic of the code, is also integrated and included as part of the Integrated Development Environment. Program flow diagrams are comprised of program block icons and arrows to depict the code's program flow. Data flow diagrams are comprised of icons depicting data processing steps and arrows to depict the flow of the data through the program. Preferably, the visualizer and editor are integrated so that changes made to the code in the editor are immediately reflected in the visualizer and vice-versa. The visualizer can also read information from execution logs and execution outputs to display the execution path for selected code and automatically display insightful debugging and optimization information for the selected code to the user.

To assist in developing new code or editing existing code, the Integrated Development Environment further includes a template manager that allows the user to browse through a repository of existing code or templates and to copy templates into the selected code for editing.

For allowing the editor to process code that is written in different Data Manipulation System programming languages and for creating the program flow icons, the Integrated Development Environment additionally includes a parser layer. The parser layer detects the type of code in the selected file and activates the rules and logic that apply to the corresponding Data Manipulation System programming language.

As will become apparent from the detailed description that follows, the subject Integrated Development Environment provides, among others, the following unique functions: seamlessly exchanging with and executing files on local and remote computers, where the site manager is

capable of compiling connection information for remote computers necessary to achieve the seamless transfer and execution of files; automatically generating program flow and data flow diagrams, where the program flow and data flow diagrams can be viewed at various levels of abstraction and where the user is capable of utilizing a step-wise function to collapse or expand the levels of abstraction to view; and automatically parsing the execution log to automatically match errors and warnings in the log file to the appropriate corresponding lines of code in the program file in order to ease the ability to correct the error, to visually highlight problematic areas, and to generate user customizable error messages and debugging advice for such problematic areas.

A better understanding of these and other objects, advantages, features, properties and relationships of the invention will be obtained from the following detailed description and accompanying drawings which set forth an illustrative embodiment and which are indicative of the various ways in which the principles of the invention may be employed.

BRIEF DESCRIPTION OF DRAWINGS

For a better understanding of the invention, reference may be had to a preferred embodiment shown in the following drawings in which:

FIG. **1** is a diagram illustrating an exemplary computer network for generating, maintaining and executing computer code;

FIG. **2** is a diagram illustrating exemplary components of the local computer;

FIG. **3** is an exemplary screen shot depicting a graphical user interface displaying a Menu bar, a Tool bar, a Display area (in this case the Editor) and a Navigation bar;

FIG. **4***a–f* are exemplary screen shots and corresponding tables depicting icons that are representative of menu items included on the Menu bar, along with the name of each menu item and a functional description of pertinent menu items;

FIG. **5** is a diagram illustrating the tool bar and a corresponding table describing the pertinent functions associated with the buttons included on the Tool bar;

FIG. **6** is an exemplary screen shot depicting an Editor window in Full Screen mode, along with corresponding output and log file tabs;

FIG. **7** is an exemplary screen shot depicting a tree view, along with the corresponding code segments;

FIG. **8** is an exemplary screen shot depicting a template manager window with available templates;

FIG. **9** is an exemplary screen shot depicting a program flow for a selected file, along with arrows that indicate the flow of data within the program flow;

FIG. **10** is an exemplary screen shot depicting a server module window configured for automated login and including session tabs for Server A and Server B;

FIG. **11** is an exemplary screen shot depicting a site manager window;

FIG. **12** is an exemplary screen shot depicting a document manager window;

FIG. **13** is an exemplary screen shot depicting a search panel for locating files;

FIG. **14***a* is an exemplary screen shot depicting an Enhanced Editor Options window;

FIG. **14***b* is an exemplary screen shot depicting a Color tab window for customizing the font colors associated with the code displayed by the Editor;

FIG. **14***c* is an exemplary screen shot depicting a General tab window for configuring the usage of an external editor in-lieu of the built-in editor, enabling and disabling line numbering for the code displayed by the Editor, and selecting print options;

FIGS. **14***d&e* are exemplary screen shots depicting an Execution Configuration tab for configuring the execution mode and execution location for selected files;

FIG. **15** is an exemplary screen shot depicting the template window as it is displaying available web-based templates to the user;

FIG. **16** is an exemplary screen shot of a web-based template that is already configured;

FIG. **17** is an exemplary screen shot depicting a data flow for a selected file;

FIG. **18***a–c* are a series of exemplary screen shots depicting operation of a step-wise function and various data flows as the data flows are being collapsed;

FIG. **19** is an exemplary screen shot depicting the visualizer employing the Split Screen view;

FIG. **20** is an exemplary screen shot depicting the visualizer with a problematic code section and a corresponding program flow icon being displayed in red;

FIG. **20***a* is an exemplary screen shot depicting the visualizer with a problematic code section and a corresponding error log for that code section;

FIG. **20***b* is an exemplary screen shot depicting a debugging hint associated with a problematic code section;

FIG. **20***c* is an exemplary screen shot depicting an error message and a corresponding debugging hint being provided by the message manager;

FIG. **21** depicts a predefined class structure for recognizing and displaying tokens, which is employed by a file parser to parse a selected file; and

FIG. **22** depicts an exemplary file that has been parsed and the corresponding class structure of the parsed file.

FIG. **23** depicts an exemplary file with break points added.

DETAILED DESCRIPTION

Turning now to the Figures, wherein like reference numerals refer to like elements, there is illustrated an Integrated Development Environment having numerous cooperating modules which together provide a system and method for generating and maintaining software, in particular, the software for data development and data manipulation languages. Although not required, the system and method will be described in the general context of a computer network **20**, illustrated in FIG. 1, and computer executable instructions being executed by general purpose computing devices within the computer network **20**. In this regard, the general purpose computing devices may comprise one or more remote computers **22***a*, and one or more local computers **22***b*, hosting an integrated software application **30**. The computer network **20** can also include one or more databases **24**. It should be appreciated that the network components could be described as having client and server relationships, as generally known in the art.

To allow the local computers **22***b* to generate and maintain code written in various programming languages, the integrated software application **30** will reside on the local computer **22***b*. Further, as shown in FIG. 2, it is preferable that the integrated software application **30** execute on a Java Virtual Machine ("JVM") which acts as an interface between the integrated software application and the operating system for the local computer **22***b*. Although the operating system for the local computer **22***b* is preferably Windows® based, it should be understood that the local

computer **22***b* could employ any one of the currently existing operating systems, such as LINUX®, UNIX®, MAC OS®, etc.

For editing, generating and maintaining software (i.e., program code), the local computers **22***b* include a graphical user interface **40**. As shown in FIG. **3**, the graphical user interface **40** is further comprised of a menu bar **42**, a tool bar **44**, a display area **46** and a navigation bar **48**. FIGS. **4***a*–**4***e* shows exemplary drop-down menu items included on the menu bar, along with a brief description of the functionality associated with those options. FIG. **4***f* shows the "Right-Click" menu. The tool bar **44** is further comprised of and displays several buttons, including a template manager button **44***a*, which serves as a link to a template manager **100**. In addition, FIG. **5** shows exemplary buttons included on the tool bar **44**, along with a brief description of the functionality associated with those buttons. The navigation bar **48** is further comprised of and displays a document manager button **48***a*, a site manager button **48***b*, an editor button **48***c*, a visualizer button **48***d*, a database manager button **48***f*, and a server button **48***e*, which serve as links to modules corresponding with these respective buttons, i.e., a document manager **60**, a site manager **70**, an editor **80**, a visualizer **120**, a database manager **121**, and a server module **160**, each of which will be described in greater detail below.

As will be appreciated by those of skill in the art, the computers **22***a*, **22***b* need not be limited to personal computers, but may include hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, minicomputers, mainframe computers, personal digital assistants, cellular telephones or the like depending upon their intended end use within the system. For performing the procedures described hereinafter, the computer executable instructions may be written as routines, programs, objects, components, and/or data structures that perform particular tasks. Within the computer network **20**, the computer executable instructions may reside on a single computer **22***a*, **22***b* or the tasks performed by the computer executable instructions may be distributed among a plurality of the computers **22***a*, **22***b*. Therefore, while described in the context of a computer network, it should also be understood that the present invention may be embodied in a stand-alone, general purpose computing device that need not be connected to a network.

To perform the particular tasks in accordance with the computer executable instructions, the computers **22***a*, **22***b* may include, as needed, a video adapter, a processing unit, a system memory, and a system bus that couples the system memory to the processing unit. The video adapter allows the computers **22***a*, **22***b* to support a display, such as a cathode ray tube ("CRT"), a liquid crystal display ("LCD"), a flat screen monitor, a touch screen monitor or similar means for displaying textual and graphical data to a user. The display allows a user to view information, such as, code, file directories, error logs, execution logs and graphical user interface tools.

The computers **22***a*, **22***b* may further include read only memory (ROM), a hard disk drive for reading from and writing to a hard disk, a magnetic disk drive for reading from and writing to a magnetic disk, and/or an optical disk drive for reading from and writing to a removable optical disk. The hard disk drive, magnetic disk drive, and optical disk drive may be connected to the system bus by a hard disk drive interface, a magnetic disk drive interface, and an optical disk drive interface, respectively. The drives and their associated computer-readable media provide a means of non-volatile storage for the computer executable instructions and any other data structures, program modules, databases, etc. utilized during the operation of the computers **22***a*, **22***b*.

To connect the computers **22***a*, **22***b* within the computer network **20**, the computers **22***a*, **22***b* may include a network interface or adapter. When used in a wide area network, such as the Internet, the computers **22***a*, **22***b* typically include a modem or similar device. The modem, which may be internal or external, is connected to the system bus via a serial port interface. It will be appreciated that the described network connections are exemplary and that other means of establishing a communications link between the computers **22***a*, **22***b* may be used. For example, the system may also include a wireless access interface that receives and transmits information via a wireless communications medium, such as a cellular communications network, a satellite communications network, or another similar type of wireless network. It should also be appreciated that the network interface will be capable of employing TCP/IP, FTP, SFTP, Telnet SSH, HTTP, SHTTP, RSH, REXEC, etc. and other network connectivity protocols.

For seamlessly transferring files, the document manager **60** is utilized. The document manager **60** is a file management program that performs many enhanced file management functions, such as recognizing related files, (e.g., execution log files, output files, include files, etc.) and managing related files as a unit, regardless of the location of such files. For example, if the user causes the document manager **60** to transfer a file from the remote computer **22***a* to the local computer **22***b*, the document manager **60** may also determine whether any related files exist, i.e., files that are or will be used in conjunction with the file that the user is transferring, and transfer those files as well. As will be described in further detail below, these enhanced file management functions, including managing related files and storing information, such as origination and timestamp information about these files, allow the user to exchange files with and execute code on local computers **22***b* and/or remote computers **22***a*. To accomplish the enhanced file management functions, the document manager **60** may also include an intelligent module **62**, a security layer **66** and a file transfer program **68**.

The intelligent module **62** enables the document manager **60** to track pertinent file transfer information **64** that enable the performance of enhanced file management functions, including, uploading files from a source computer and returning/downloading an edited version of the same file to the same source computer for storage, execution, etc. To accomplish the functions of returning edited files to the source computer and executing edited files remotely, the file transfer information **64** may include a source computer identifier, a file directory identifier, a content identifier, a timestamp or other information capable of allowing the tracking of the type or path of files that are transferred between computers **22***a*, **22***b* or within a single computer **22***b*.

To ensure that the integrity and security of the network **20** is maintained, a security layer **66** is utilized. The security layer **66**, which is part of the site manager **70**, may save server login information supplied by the site manager **70** thereby allowing subsequent file exchanges to be handled transparently. The security layer **66** may also be capable of configuring the integrated software application **30** to work with firewalls. The site manager **70** gathers server login information and similar information for use by the document manager **60**, the intelligent module **62** and the security layer **66**, and interacts closely with each of these modules. Pref-

erably, remote file access and file execution are handled centrally through the site manager **70**.

To maintain existing software and develop new software, the editor **80** allows the user to perform standard text editing functions, including, mouse placement of the cursor, click-and-drag text selection and standard Windows® key combinations for cutting, copying and pasting data. While editing code that has been previously executed, the associated log files and output files that are stored in the same directory **104** as the code will automatically be opened and each of these files will include a corresponding tab **84***a*, **84***b*, as shown in FIG. **6**. The user can access each of these files by clicking on the corresponding tab **84***a* or **84***b* and switch between several related files by clicking on different tabs. The editor **80**, the document manager **60** and the site manager **70** cooperate to track the association between edited code, the log files and output files that are generated by the edited code, and the remote system on which edited code may be executed. The editor **80** may also employ language-specific syntax checking and auto-correct functions to enhance the software development capability of the integrated software application **30**.

To make the code more manageable and allow the user to see a more abstract version of the code, the editor **80** also provides a means to invoke a tree view **90**, which may be displayed near the editor **80**. The tree view **90** depicts individual procedures and data blocks as active elements **82** and shows the code at a high level. Since each active element **82** is representative of a larger code segment, as shown in FIG. **7**, the user can navigate throughout the code by selecting active elements **82** and thereby displaying the larger code segment associated with the selected active element **82** (i.e., the editor will automatically scroll to the region where the code segment is visible). To make it easier to cut, copy or move code segment, the user can right-click on an active elements in the tree. This will cause the editor to automatically scroll to the region where the code segment is located and highlight the entire code segment. The user can then cut or copy the highlighted code segment to the clipboard.

For enabling users to add pre-existing lines of code or templates **106** to a program being edited or created, the editor **80** also includes the template manager **110**, which may be accessible to the user by clicking on the Template button on the tool bar **44** menu. As shown in FIG. **8**, the template manager window **102** includes a directory **104** of templates **106**, which are both pre-built and written by the user and which may be examined by navigating the directory **104** in a manner that is well-known in the art. The template manager **110** allows the user to insert templates **106** directly into the code that is being created/edited. In addition, the template manager **110** also allows the user to create and organize their own templates **106**. Still further, the template manager **110** could be used in connection with the Internet and a web browser to view and retrieve templates stored at remote locations, such as web sites, bulletin boards, etc. The Template Manager **100** can store the templates (both built-in and those developed by the user) locally and/or centrally on a remote server making it easy for the entire workgroup or enterprise to share these templates.

In addition, the editor **80** may also provide an auto-complete function capable of automatically generating code based on templates **106** managed by the template manager **100**. For example, as soon as a user keys in a recognized key word, the editor **80** will automatically perform a look-up function for that keyword in the directory **104**. If a template **106** includes a matching keyword, then the editor **80** may automatically paste the template into the display area. If more than one template **106** matches the keyword, then a pop-up window will be displayed presenting the user with the option of selecting one of the matching templates. It should also be appreciated that the user customize the integrated software application **30** by enabling or disabling the auto-complete feature.

For viewing the program flow and data flow of a selected program, as will be further described below, the visualizer **120**, in connection with a parser layer **140**, reads, parses and displays the code for the selected program, representing each program and data block with a program flow icon **126**. As illustrated in FIG. **9**, arrows connect these program flow icons **126** to generally illustrate the flow of data.

Assuming that the designated code segment has been executed previously and that the execution log is readily available, the user may also "mouse-over" the program flow icon to display the comments and execution statistics, i.e., CPU usage, number of row processed, etc., associated with the block. Right-clicking on the program flow icon will activate specific functions or options associated with the icon. For example, the dataset icon (i.e., an icon that represents a dataset or a table) will display a list of data inspection or data discovery functions, including ad hoc SQL® queries. Ad hoc SQL® queries that are selected may be executed by the respective engine. These engines (e.g., SAS, DB2 UDB, or other RDBMS servers) can resides locally on the user's workstation or remotely on the LAN/WAN and the results, once retrieved will be displayed in a pop-up output window

To access and interact with remote computers **22***a* via command lines, the server module **160** acts as a robust terminal emulator. The server module **160** allows users to open one or more sessions thereby simultaneously gaining access to one or more hosts/remote computers **22***a*. In addition, the user has the option of executing edited code on a remote computer **22***a*, by employing the server module **160**, in connection with the site manager **70**, to connect the local computer **22***b* to the remote computer **22***a*, as will be described in more detail below. Once a session is opened, a terminal tab for the respective session can be created and displayed to the user by the server module **160**, as illustrated in FIG. **10**. It should be understood by those with skill in the art that the server module **160**, the site manager **70** and the document manager **60** all preferably interact with one another to effectuate the transfer of code between the remote server computers **22***a* and the local computer **22***b*. It should also be understood that each of these modules could be combined or further divided to form one single module or additional modules.

The site manager **70** assists the local computer **22***b* with access to the remote computers **22***a*. For example, the document manager **60** and the server module **160** use the site manager **70** to collect the connection information **78** necessary for the local computer **22***b* to make a LAN/WAN connection to a remote computer **22***a* using either FTP, SFTP, Telnet SSH, HTTP, SHTTP, RSH, REXEC and other TCP/IP connectivity protocols. Additionally, to avoid manually entering the connection information **78** each time a different remote computer **22***a* is accessed, the site manager **70** is also capable of saving connection information **78** for different servers. Thus, the site manager **70** creates a virtual computing environment by expanding the computing boundary of the local computer **22***b* to include remote computers **22***a* and making various computing resources across the LAN/WAN seamlessly available for use by the local computer **22***b*. As shown in FIG. **11**, the site manager

window **72** may be divided into a first panel **74** and a second panel **76**. The first panel **74** displays a tree structure that depicts the system configuration **74***a* for each of the remote computers **22***a* that are available to the local computer **22***b*. The second panel **76** displays the connection information **78** for a selected remote computer **22***a*, including but not limited to the following fields: Profile Name **78***a*, Host Address **78***b*, User **78***c*, Password **78***d*, FTP port **78***e* and Telnet Port **78***f*.

The site manager **70** also includes a PASV mode and Firewall option which can be selected by the user. PASV mode is supported as an option to allow file transfer programs ("FTPs") to work with a firewall. This feature reverses the connection between the remote computer **22***a* and the local computer **22***b* allowing many users whose sessions are hosted behind firewalls to use the document manager **60**. Therefore, PASV mode is useful for certain types of firewalls that do not allow FTP servers to initiate data connections back to the connected client. If the Firewall option is selected, the firewall configuration parameters from the Firewall tab of the Option Dialog will be used to initiate a connection to this profile. Moreover, if the local computer **22***b* resides behind a firewall that limits or restricts FTP access and the Firewall mode is selected, the necessary connection information **78** and Firewall selections for the local computer **22***b* must be entered, i.e., a host name, port number, user ID and password. In accordance with procedures that are generally known in the art, the Firewall Type box permits the user to select the command required by the firewall to initiate an FTP session from the site manager **70**. In addition to providing options that enhance the network connectivity of the local computer **22***b*, the Option button also provides tools and wizards for automatically searching the LAN/WAN for designated resources. These tools and wizards simplify the network/server connection setup between the local computer **22***b* and the remote computers **22***a*.

For parsing the code, the parser layer **140** is provided. The file parser **142** retrieves code utilizing the services provided by the document manager **60** and examines the words/tokens which comprise the code **144**. The file parser **142** also identifies and tags the tokens **144** to indicate that they are one of a variety of token classes, such as, a keyword token, a newline token, a quotestring token, a macro token and a comment token. Since the integrated software application **30** is language aware, the parser layer **140** may accommodate varying language types. Thus, although the preferred embodiment of the present invention is specifically designed to include modules for parsing SASS® and SQL code, it should be appreciated that the modules can be included for parsing and interpreting other language types, such as, SPSS®, DB2 UDB® Store Procedures, ORACLE® PL/SQL, etc.

For managing and interacting with database files, a database manager **220** may be provided which automatically interacts with the local ODBC registry to display a list of ODBC resources. In addition, the database manager **220** may automatically search the LAN/WAN for RDBMS servers and display the list of RDBMS servers that are found locally or remotely. The database manager also allows the user to select and drill down to a desired database and automatically connect to the selected database, unless a login screen is required to connect to the database. Once connected to the selected database, the Database Manager will display a list of tables which are stored and available in the database. The user can then right-click on a particular table to gather statistics or to retrieve sample data from those tables. The user can also drill down on a particular table to display the columns of the tables. Similarly, the user can click on a particular table to gather statistics (e.g., frequency counts, min max, distinct value, average, etc.) or to retrieve sample data from those tables. The user can also drag and drop tables onto a work area to graphically generate SQL® statement, to retrieve data from the database. Additional data manipulation functions, e.g., pivot, crosstab, and templates will be available to help users inspect, transform, import, export, map, format, transport and derive new data based on existing data. The database manager **220** may also include a meta data manager **222** to help the user document the business or logical definition of the data, and the technical or physical definition of the data. The meta data manager **222** may also assist the user in tracking the lineage of the data, i.e., the source of the data, the changes to the data that were made and the destination of the data.

To further exemplify the functionality of the integrated software application **30**, the following section will discuss an exemplary software development session, along with the modules used to effectuate the respective functions.

This hypothetical session commences with activation of the integrate software application **30**. Once the integrated software application **30** is launched, the graphical user interface **40** will be displayed and the user will have the opportunity to open modules in the navigation bar **48** or engage functions represented by buttons in the menu bar **42** or the tool bar **44**. For determining to which computer the user desires access, the user would begin by activating the site manager **70**, which causes the site manager window **72** to be displayed to the user. The site manager window **72** includes a first panel **74** and a second panel **76**. As shown in FIG. **11**, the first panel **74** of the site manager window **72** displays the defined/available remote computers **22***a* for the user's selection. As discussed above, the second panel **76** of the site manager window **72** displays a server profile for a selected remote computer **22***a*, comprising connection information **78**, including, but not limited to, the following fields: Profile Name **78***a*, Host Address **78***b*, User, Password, FTP port **78***e* and Telnet Port **78***f*. As will be appreciated, the site manager can be configured to search the network for servers or certain types of servers and display the found servers for selection by the user.

The server profiles **170** may be populated by selecting a folder icon **172** to use as a parent directory **174**. Once a parent directory **174** is opened, the user may click on the New button **176** to create a new server profile **170**. Therefore, it should also be understood that the site manager **70** allows the user to create new server profiles **170** by entering the necessary connection information **78**. To edit an existing server profile **170**, the user must select a remote computer **22***a* and click the Edit button **178**. After the Edit button **178** is selected, the cursor may be moved to the Profile Name **78***a* for editing or modification, or the user may click on the Option button for Advance Network/Server Connectivity Configuration Settings.

The text displayed in the Profile Name **78***a* field can be a profile name **78***a* that was selected in the first panel **74** of the site manager window **72** or text the user enters in creating a profile for a remote computer **22***a* or host. The Host Address **78***b* field may contain an IP address or a resolvable DNS host name of the FTP, SFTP, Telnet, SSH, REXEC, HTTP, or SHTTP server for the respective host definition. The User **78***c* field may contain the login name that the user will enter to access the remote server computer FTP account. If the selected remote computer **22***a* accepts anonymous FTP requests, "anonymous" may be entered in this field. The

Password **78***d* field may contain a password **78***d* for the remote server computer FTP account, unless an anonymous login is used; then, the Password **78***d* field may remain blank. The FTP port **78***e* field is set by default to **21**, since most server computers **22***a* accept FTP connection requests on port **21**, however, the FTP Port **78***e* field may be changed as needed. The Telnet Port **78***f* field is set by default to **23**, since most server computers **22***a* accept Telnet connection requests on port **23**, however, the Telnet Port **78***f* field may also be changed as needed.

Once the user selects a remote computer **22***a* to access, the connection information **78** for the selected remote computer **22***a* is returned by the site manager **70** and displayed to the user. Then, the user is given the option of connecting to the selected remote computer **22***a* by selecting the Connect Telnet button. When the Connect Telnet button is selected, the server module **160** is activated. To connect to the remote computer **22***a*, the server module **160** utilizes the connectivity layer **180**. More specifically, as is known in the art, the connectivity layer **180** utilizes the connection information **78** with TCP/IP and other similar networking protocols to interconnect with the remote computer **22***a*. Once the server module **160** has made an initial connection with the selected remote computer **22***a*, a server module window **162** is displayed to the user. The server module window **162** displays the name of the remote computer **22***a* to which the user is trying to connect.

Prior to opening a session on a particular host, the user may be required to login or the server module **160** may automatically log the user on (depending on the settings that the user has selected in the server profile **170**, i.e., save login and password). FIG. **10** shows a server module window **162** that is configured for automated login, with a home path preset to the integrated software application's SAS® directory. The combination of the site manager **70**, the server module **160** and the connectivity layer **170**, enable the user to seamlessly access SAS® documents, log files and output files stored at the remote computer **22***b*.

Once the local computer **22***b* accesses a remote computer **22***a* using the combination of the site manager **70**, the server module **160** and the connectivity layer **170**, the document manager **60** will be activated automatically. Moreover, to view the available files for a selected computer, the document manager window **170** may be displayed to the user. As shown in FIG. **12**, the document manager window **170** includes a local panel **172** and a remote panel **174**. Each of these panels, **172**, **174** displays the available files for the respective computers **22***b*, **22***a* as a file tree structure, as shown in FIG. **12**. The user may then select one or more of the available files for editing or visualizing. In the preferred embodiment of the present invention, files that are located in the local panel **172** are resident on the local computer **22***b* and files that are located in the remote panel **174** are resident on the remote computer **22***a*, but it should be appreciated that such designations are for explanatory purposes only and that other variations of network systems and computer designations could be employed.

The document manager **60** also allows the local computer **22***b* to perform various Windows® commands, such as, creating new files, renaming, deleting and opening existing files, printing files, copying, cutting and pasting information to the Windows® clipboard, and other standard commands that are generally known in the art. In addition, the document manager **60** provides a Search Option **69**. As shown in FIG. **13**, selecting the Search Option **69** opens the search option window **69***a* and allows users to search any file system that is accessible to the local workstation/local computer **22***b*, in accordance with generally known file searching techniques, including by exact or partial file name, modification date or date range, and content.

To open a file displayed in the local panel **172**, the user may double-click on the selected file, or highlight the selected file and start the desired module (i.e., the editor **80** or the visualizer **120**). If the user double-clicks on a file located in the local panel **172**, the editor **80** will automatically open and display the file. To open and edit a file displayed in the remote panel **174**, it is preferable that the user drag a selected file from the remote panel **174** of the document manager **60** and drop the selected file in the local panel **172** of the document manager **60**. It should also be understood by those with skill in the art that other methods for opening and moving files may be employed.

To retrieve a file that is stored on a remote computer **22***a*, the document manager **60** may utilize a file transfer program **64**, in connection with the site manager **70** and the connectivity layer **180**. For example, as is known in the art, the document manager **60** may send a request to the file transfer program **64** to import the selected file from a particular remote computer **22***a*. In response to this request, the file transfer program **64** may communicate with the remote computer **22***a* via the connectivity layer **180** and instruct the remote computer **22***a* to send the selected file to the local computer **22***b*. When the selected file is being transferred from the remote computer **22***a* to the local computer **22***b*, the intelligent module **62** of the document manager **60** may compile file transfer information **64**, as noted above. It should also be understood that, while it is preferred to have the intelligent module **62** compile file transfer information **64**, other similar modules may be equally capable of accomplishing the task of compiling file transfer information **64** for the system **20**.

Again, for exemplary purposes only, it is assumed that the user has chosen to edit the code before visualizing it. As mentioned above, to begin editing files, the user may either double-click on the selected file, which automatically opens the editor **80**, or the user may highlight the selected file and then open the editor **80**. To open the editor **80**, the user need only double-click on the Editor button **48***c* located in the navigation bar **48** of the graphical user interface **40**, as shown in FIG. **3**.

As shown in FIG. **6**, once the editor **80** begins the editing process (the default setting is Full Screen mode), the contents of a selected file are returned from the document manager **60** and displayed to the user beside a navigation bar **48** bar. For editing the code and implementing advanced code-handling capabilities, the editor **80** also includes features, such as, language-aware syntax highlighting, warning and error log file highlighting, automatic line numbering, automatic completion of code segment based on program templates, tree views of program blocks and code templates **106**. By highlighting key code statements, the language-aware syntax highlighting makes the code easier to read and edit. In addition, by highlighting the code, based on information contained within the warning and error log files, errors in the code are more apparent and the code is easier to trouble-shoot.

To allow the user to customize, edit and maintain error messages and the corresponding debugging hints, the message manager **88** is provided. More specifically, the message manager **88** allows the user to create, update and delete error messages. For example, the user can click on any error message in the log and in response to this selection, the message manager **88** may compare the selected error message to an existing repository of error messages The reposi-

tory of error messages can be stored locally and/or centrally in a remote server accessible and sharable by the entire workgroup or the entire enterprise. If the selected error message matches one of the existing error messages, a debugging window will pop-up. The debugging window will display the original error message, along with any corresponding debugging hints that are associated with the original error message as provided by the message manager **88**. If the selected error message does not match one of the existing error messages, the message manager **88** may automatically read the new message into the repository of error messages. Additionally, the user may have the option to associate debugging hints with the new error message.

To further add to the utility of the editor **80**, the user can open the Enhanced Editor Options window **84** by clicking the Editor Configuration button **86** in the Options pull-down menu. The Enhanced Editor Options window **84** is further comprised of a General tab **84***a*, a Font tab **84***b*, a Color tab **84***c* and an Execution tab **84***d*. As is generally understood in the art and as shown in FIG. **14***a*, the font type, size and style of the code may also be customized by the user by selecting the Font tab **84***b* in the Enhanced Editor Option window **84**. Additionally, as shown in FIG. **14***b*, the Color tab **84***c* of the Enhanced Editor Options window **84** may be selected to allow the user to select the colors that are used in connection with various syntactically significant language elements in the SAS® program file, thereby allowing the user to customize the language-aware syntax highlighting and warning and error log file highlighting functions. To further configure the user workspace, the editor **80** allows the user to choose whether or not to display line numbers in connection with editing the code. For example, as shown in FIG. **14***c*, the General tab **84***a* can be selected by the user and may provide options for selecting a line numbering button representative of disabling or enabling the line numbering, specifying printing options or indicating the use of alternate editor. For allowing the integrated software application **30** to configure the execution path for selected files, the Execution tab **84***d* can be selected by the user and the execution mode and execution location may be defined (i.e., Locally or Remotely), as illustrated in FIGS. **14***d*&*e*. By configuring the execution path for selected files, the user can execute the code on local or remote computers **22***b*, **22***a* or engines.

Once the selected code is displayed on the editor window **82**, the user can begin to edit the code. Although the default setting for the editor window **82** is the Full Screen mode, the user can also select a Split Screen mode. The Split Screen mode splits the editor window **82** into two panels, i.e., the tree view panel **82***a* and the code view panel **82***b*. To enhance the ability of the user to navigate within the body of the code, the tree view panel **82***a* displays the tree view **90**, which is comprised of a group of active elements **92**. The tree view panel **82***a* displays a high-level representation of the code and the code view panel **82***b* displays the corresponding code that is associated with a highlighted active element **92**. To this end, the editor **80** works in connection with the parser layer **140** to parse the code in a manner that allows the editor **80** to display a tree view **90** comprised of active elements **92** that are representative of code section that perform important functions, such as, the introduction of new variables or the execution of append, print or similar functions. Thus, the user can navigate through the code by clicking on the desired active element **92**, which causes the code that corresponds with that active element to be displayed in the code view panel **82***b*.

The user can also add pre-existing code to the program being edited, by opening the template manager **100**. The template manager **100** allows the user to select existing templates **106** and to integrate the code associated with those templates **106** into the code by simply clicking on the paste button or by utilizing standard cut and paste tools, or similar functions, as will be explained in greater detail below.

Once the template button **44***a* is selected, the template manager **100** is activated and the template manager window **101** is opened. It should be appreciated that the template manager **100** may be a separate browser; therefore, the template manager window **101** may remain open throughout the users session. The template manager window **101** may be further comprised of a template directory panel **101***a* and a template code panel **101***b*. The template directory panel **101***a* may display a directory **104** of template folders **105** or available templates **106**. To display the code associated with an available template **106**, the user may select a file folder containing available templates **106** from the directory **104** and select an available template **106**, which is located within the selected file folder. Once a template **106** is selected, the template code **106***a* associated with the template **106** is returned to the template manager **100** and displayed in the template code panel **101***b*.

To insert the template code **106***a* associated with a selected template **106** directly into the code that is being edited, the user may click the Copy button **102** on the template manager window **101**; this places the template code **106***a* associated with the selected template **106** into the Windows® clipboard. Then, the user should place the cursor in the desired location within the code and select the Paste button **103**. Or the user may simply click the paste button and the template will be copied directly into the current location of the cursor in the document.

The template manager **100** may also allow the user to create new template folders **105** for storing templates **106**, to create new templates **106** and to edit existing templates **106**. These templates can reside on either the local drive or the network drive. In addition, for allowing the template manager **100** to import additional templates **106** from remote computers **22***a*, the template manager **100** also enables the user to browse web-based templates **108**. As shown in FIG. **15**, once the web-based templates **108** are accessed and displayed on the template manager window **101**, the web-based templates **108** may be utilized in a manner similar to the templates **106**. As shown in FIG. **16**, additional web-based templates **108** may be added to the template manager **100** by the user. It should also be understood that the templates **106** are web enabled, i.e., the user can associate links or URLs with the template during creation. Thus, users may be capable of clicking on a link and causing the template manager **100** to automatically activate a web browser to display any content associated with the link.

Once the user has completed the editing or development of the code, the user can execute and debug the code. In addition, the code can be executed locally or remotely. As is known in the art, the user may also use the editor **80** to set break points **112** in the code. As shown in FIG. **23**, break points **112** can be set by selecting the line numbers that correspond to the intended location of the break points **112**. It should also be appreciated that by setting break points **112** in the code, the code can be executed in its entirety or as a block of code with the additional option of skipping specified code segments within the selected blocks.

The site manager **70**, the document manager **60** and the editor **80** work in conjunction with one another to execute the code. For example, the site manager **70** manages the connections between the local computer **22***b* and the remote host/remote computer **22***a*; the document manager **60** tracks

the associations between the code, the log files and the output files that are generated and the remote system on which the code is to be executed; and the editor **80** allows the user to modify the code. Moreover, for assisting users in developing and maintaining software, the editor **80** and the visualizer **120** have an integrated relationship. For example, each of the active elements **92** displayed in the tree view **90** may also be represented in the visualizer **120** and any changes made in the editor **80** can be simultaneously reflected in the visualizer **120**.

To execute code remotely when the code resides on a remote computer **22***a*, the code is first copied to a directory on the local computer **22***b*. Next, the user should open the local copy of the code. After the local copy is opened and returned to the editor **80**, the code view panel **82***b* will display the code. The user can also use the document property wizard to specify the server and the location in which they intend to execute the code. The server or location may be selected by browsing the server list in the site manager **70**.

To execute code remotely when the code resides on a local workstation/local computer **22***b*, the user first instructs the site manager **70** to make a connection to the remote computer **22***a*. Then, the user may cause the document manager **60** to copy the code from the local computer **22***b* to a directory **104** on the remote computer **22***a*. To select code for execution, the user can drag an icon representing the selected code from the local panel **172** of the document manager **60** window and drop the icon in the remote panel **174** of the document manager **60** window. It should also be appreciated that the selected code could be copied by other means that are generally known in the art. Next, the user should open the local copy of the code. After the local copy is opened, the editor window **82** will display the code.

To execute the code or code segments, the user selects the Execute Program button **44***a* on the tool bar **44** or in the Right-Click menu. Once the Execute Program button **44***a* is selected, the server module **160** connects to the proper computer **22***a*, **22***b* using the connectivity layer **180** and causes the execution script, which may be defined in the site manager **70**, to operate on the selected code. After execution of the code is complete, a system prompt will be displayed. The user may then type "exit" to return to the editor **80** window; the document manager **60** automatically transfers the session's .log an .lst files to the same directory **104** from which the code was copied and displays them as separate tabs on the Editor **80** window. The user can also configure the remote execution function to that it will exit automatically.

As shown in FIGS. **9** and **17**, the visualizer **120** may be used to show the program flow **122** or the data flow **124** of the selected code on the visualizer window **121**. Additionally, the user will be able to toggle between the program flow display **122***a* and the data flow display **124***a* by selecting the View Program Flow button **122***b* or View Data Flow button **124***b*, respectively. As shown in FIG. **9**, the program flow **122** is displayed as a default and displays program flow icons **126**, which are graphical representations of code sections, in the order that they occur in the code.

For generating the program flow icons **126**, a document view engine **200** is provided. The document view engine **200** operates in conjunction with the parser layer **140** to parse the code. Using information provided by the parser layer **140**, the document view engine can intelligently recognize and arrange individual procedures and data blocks on the visualizer window **121** and represent the procedures and data blocks as program flow icons **126**. The document view engine **200** may also allow users to assign meaning and attributes to tokens **144**, which are identified by the parser layer **140**. By assigning meanings and attributes to tokens **144**, the document view engine **200** allows the visualizer to create program flows **122** and data flows **124**.

As shown in FIG. **17**, the visualizer **120** may also show the data flow **124** of the subject code. The data flow **124** is generated by parsing the code, tracing the flow of the data through the code and displaying individual processes and data blocks in separate columns with arrows that connect the program flow icons **126** and indicate the direction of the data flow. In addition, as shown in FIG. **18**, the data flow visualizer also allows the user to combine flows in a step-wise manner, thereby following the flow of the data from start to finish. As the user clicks the Step-wise button **121***a* located on the visualizer window **121**, shown in FIG. **17**, the visualizer **120** and the document view engine **200** may re-generate the data flow **124***a* on the visualizer window **121** and collapse the number of program flow icons **126** that comprise the data flow **124**, ultimately showing the entire lineage of the data (i.e., where the data came from, how the data has been processed and where the data was stored). Thus, the visualizer **120** enables the user to view a representation of the flow of data during execution of selected code one step or program statement at a time. It should also be appreciated that the user may take a single data flow **124** and reverse the step-wise function, thereby expanding the number of program flow icons **126** that comprise the data flow **124** and showing the data flow **124** for individual sections or blocks of code.

It should also be appreciated that the integrated software application **30** allows changes to the code to be made textually or visually, i.e., by using the editor **80** or the visualizer **120**, respectively. Editing, creating or developing new code visually using the visualizer is achieved by reverse engineering the exiting code or code templates, i.e., SAS®, SQL®, SPSS®, DB2 UDB®, Oracle® RDBMS and UNIX® Scripts, and displaying the code visually using icons. As the user manipulate these icons visually, code will be generated. Any changes to the code via the visual interface can be forward engineered to assume a textual format capable of being executed on the respective Data Development and Data Management System. Therefore, the integrated software application **30** is capable of producing a textual file that is derived from a visual model and executing the derived textual file.

As shown in FIG. **19**, the visualizer window **121** may also be configured to display a split view. When the visualizer window **121** is in split view mode, the visualizer window **121** will be comprised of a flow panel **121***b* and an visualizer code panel **121***c*. The flow panel **121***b* will display either the program flow **122** or the data flow **124**, and the visualizer code panel **121***c* will display the code for program flow icons **126**, which are shown in the flow panel **121***b*. Moreover, by clicking on an icon in the program flow window or data flow window, the user can cause the editor **80** window **100** to display the portion of the code that corresponds to the program flow icon **126** that was selected. This is especially helpful in debugging the code.

For example, as shown in FIG. **20**, when elements appear in red, this signifies that the highlighted element contains an error in the underlying code. If the user selects the highlighted element, a pop-up window may be displayed to the user depicting the code associated with the highlighted element, along with the corresponding Error log, i.e., the editor **80** will scroll to the corresponding section of code and display the code to the user, while also displaying the Error

log in close proximity to the code. It should be appreciated by those with skill in the art that during the user session, the user may freely navigate between the document manager **60**, the editor **80** and the visualizer **120**, as needed, by selecting the corresponding icon in the navigation bar **48**.

It should also be understood that the user can visualize the execution log. Visualizing the execution log will show the exact path of the program flow **122** and data flow **124**. The program flow **122** and the data flow **124** will be exact because they are based on the actual execution of the code. This in turns provides additional debugging and optimization information, such as, the code section that will get executed, how much data is being processed, the execution time for the code, the external files, library or macros that are referenced by the code, the format for the fully instantiated macro, etc.

For parsing the code, the document manager **60** first determines the file type for a selected file, i.e. the SAS®, SPSS®, SQL®, DB2 UDB®, Oracle® RDBMS etc. After the file type for a selected portion of code is determined, the parser layer **140** deploys the corresponding file parser **142**, e.g., a file parser **142** that corresponds to, in this case, one of a variety of data manipulation and/or data management programming languages. By deploying the appropriate file parser **142**, the parser layer **140** also activates the respective rules and logic that correspond to the detected programming language. Therefore, users are capable of developing, editing and maintaining code that can be executed by more than one data manipulation and/or data management program.

As mentioned earlier, the parser layer **140** is capable of processing varying file types, as the integrated software application **30** has been designed to be language aware. For example, as mentioned above, the document manager **60** recognizes file types and the integrated software application **30** includes enhanced, standalone productivity tools, such as, generic text editors, the Windows File Manager and File Transfer programs **68**. In addition, these productivity tools are also designed to work seamlessly with the integrated software application **30** by implementing XML protocols, as is generally known in the art. Thus, although the preferred embodiment of the present invention is designed to interact with SAS® code, it should also be appreciated that the system will also be capable of parsing and interpreting other file types, such as, SPSS®, SQL®, DB2 UDB®, Oracle® RDBMS, etc.

As is generally known in the art, after the file parser **142** retrieves code from the document manager **60**, the file parser **142** breaks the code document **146** into individual words/tokens **144**. Based on the class of the individual tokens **144**, the file parser **142** identifies and tags the tokens **144**. Tokens **144** can be tagged to indicate that they are one of a variety of classes, such as, a keyword token **144***a*, a newline token **144***b*, a quotestring token **144***c*, a macro token **144***d* and a comment token **144***e*. By tagging the tokens **144**, the parser layer **140** enables the document view engine **200** to recognize and display the program flow **122**. As shown in FIG. **21**, the document view engine **200** employs a predefined class structure for recognizing and displaying the tokens **144** provided by the file parser **142**. An example of the class structure as it might be implemented in a parsed document/file is shown in FIG. **22**.

While specific embodiments of the present invention have been described in detail, it will be appreciated by those skilled in the art that various modifications and alternatives to those details could be developed in light of the overall teachings of the disclosure. For example, the processes described with respect to computer executable instructions can be performed in hardware or software without departing from the spirit of the invention. Furthermore, the order of all steps disclosed in the figures and discussed above has been provided for exemplary purposes only. Therefore, it should be understood by those skilled in the art that these steps may be rearranged and altered without departing from the spirit of the present invention. In addition, it is to be understood that all patents discussed in this document are to be incorporated herein by reference in their entirety. Moreover, while the present invention may be described in terms of a particular programming language, it should also be understood that the present invention may be programmed in various other software languages. Accordingly, the particular arrangement disclosed is meant to be illustrative only and not limiting as to the scope of the invention which is to be given the full breadth of the appended claims and any equivalents thereof.

What is claimed is:

1. An integrated development environment, comprising:

a document manager for retrieving source code programmed using one of a plurality of types of data manipulation languages;

an editor for displaying the retrieved source code and providing a means for a user to edit the retrieved source code;

a parser layer which detects the one of the plurality of types of data manipulation languages in which the retrieved source code is programmed and which activates rules and logic applicable to the detected one of the plurality of types of data manipulation languages; and

a visualizer dynamically linked to the editor for displaying graphical representations of flows within the retrieved source code using the rules and logic applicable to the detected one of the plurality of types of data manipulation languages and activated by the parser, wherein the editor, parser layer and visualizer cooperate such that edits made to the source code using the editor are automatically reflected in the graphical representations of flows displayed by the visualizer and edits made to the graphical representations of flows in the visualizer are automatically reflected in the source code displayed by the editor.

2. The integrated development environment as recited in claim **1**, wherein the graphical representations of flows depict data flows.

3. The integrated development environment as recited in claim **1**, wherein the graphical representations of flows depict program flows.

4. The integrated development environment as recited in claim **1**, wherein the graphical representations of data flows are expandable and collapsible.

5. The integrated development environment as recited in claim **1**, wherein the document manager retrieves all files related to the source code to be edited.

6. The integrated development environment as recited in claim **1**, wherein the document manager comprises a site manager and a connectivity layer for retrieving source code from one or more remote computers.

7. The integrated development environment as recited in claim **6**, wherein the document manager comprises a security layer for managing secure connections with the one or more remote computers.

8. The integrated development environment as recited in claim **1**, wherein the editor comprises a template manager for allowing preprogrammed segment of source code to be placed within the source code being edited.

9. The integrated development environment as recited in claim 8, wherein the template manager is adapted to automatically correct segments of the source code.

10. The integrated development environment as recited in claim 8, wherein the template manager is adapted to automatically generate segments of the source code.

11. The integrated development environment as recited in claim 1, further comprising a means for allowing the source code to be executed both locally and remotely.

12. The integrated development environment as recited in claim 11, wherein the parser layer further examines error log files generated by the means for allowing the source code to be executed to determine segments of the source code determined to include errors.

13. The integrated development environment as recited in claim 12, wherein the visualizer cooperates with the parser layer to change the appearance of displayed flows as a function of the source code segments determined to have errors.

14. The integrated development environment as recited in claim 12, wherein the editor cooperates with the parser layer to change the appearance of portions of the displayed source code as a function of the software segments determined to have errors.

15. The integrated development environment as recited in claim 12, further comprising a message manager cooperating with the parser layer for displaying debugging hints as a function of the source code segments determined to have errors.

16. The integrated development environment as recited in claim 15, wherein the message manager allows a user to edit and maintain debugging hints for a variety of different errors.

* * * * *